

WillowFold: Secure Aggregation with a Lightweight Committee

Hossein Hafezi
NYU & KU Leuven
h.hafezi@nyu.edu

Kasra Abbaszadeh
University of Maryland
kasraz@umd.edu

Adrià Gascón
Google
adriag@google.com

Phillipp Schoppmann
Google
schoppmann@google.com

Mariana Raykova
Google
marianar@google.com

Benedikt Bünz
NYU
bb@nyu.edu

Abstract

Secure aggregation enables a server to learn the sum of private inputs of clients, while revealing no additional information beyond the final sum. Recent work, Willow (CRYPTO 2025) achieves one-shot secure aggregation in the single-server model with dynamic client participation. Willow relies on an auxiliary committee whose verification work scales linearly with the number of clients. This linear committee cost limitation is also shared by other state-of-the-art single-server secure aggregation protocols, including Flamingo (IEEE S&P 2023) and OPA (CRYPTO 2025). This creates a deployment challenge: the verification burden falls on volunteer committee members rather than the server, which is the party that actually benefits from the aggregation. It is more natural for the server to absorb the bulk of the cost.

In this paper, we introduce **WillowFold**, a secure aggregation scheme based on Willow that enables lightweight verification of the aggregation and requires only logarithmic committee cost in the number of clients, shifting the verification burden to the server where the incentives properly lie. Our scheme additionally supports streaming aggregation and reduces per-client server storage to a single ID (4 bytes), rather than storing full contributions as in Willow (1.5 KB), which facilitates distributed execution of the server-side computation. Finally, we close a gap in Willow’s security proof by showing that the zero-knowledge proofs underlying the scheme must be simulation extractable in order to rule out correlated ciphertext attacks.

WillowFold employs proof-carrying data (PCD)—a primitive for incremental verification of distributed computations—to achieve a lightweight verification cost. In particular, we present a concretely efficient instantiation of **WillowFold** using a variant of Spartan+KZH-Fold (CCS 2025) as the underlying PCD scheme. As independent contributions, we show a zero-knowledge variant of Spartan+KZH, which is the SNARK underlying Spartan+KZH-Fold, that uses logarithmic randomness, and we further prove that this variant satisfies the simulation extractability.

We implement and evaluate **WillowFold**, showing that it supports 2^{23} clients with proof size < 60 KB and committee verification time under one second—a 10^5 -fold improvement over Willow. This reduction comes at a modest server-side cost: per-client server computation increases from 0.2 seconds in Willow to 4.3 seconds in **WillowFold**, a trade-off that aligns the computational burden with the party that benefits from the aggregation.

Contents

1	Introduction	4
1.1	Our Contributions	5
1.2	Related Work and Comparison	6
2	Single-Server Secure Aggregation	8
2.1	Roles and Architecture	9
2.2	Secure Aggregation Functionality	9
3	Technical Overview	9
3.1	WillowFold Secure Aggregation	10
3.2	Requirements of the PCD Scheme	12
3.3	Instantiation with Spartan+KZH-Fold	13
4	Preliminaries	14
4.1	Argument Systems	14
4.2	Commitment Schemes	15
4.3	Homomorphic Encryption Schemes	16
5	Secure Aggregation with a Lightweight Committee	17
5.1	Setup and Registration	17
5.2	Client, Server, and Decryptor Roles	17
6	Security Analysis	21
6.1	Ciphertext Correlation Intractability	21
6.2	Simulation Security Proofs of WillowFold	21
7	Concrete Instantiation with Spartan+KZH-Fold	23
7.1	Properties of Spartan+KZH-Fold	24
7.2	Zero-Knowledge Spartan+KZH	25
7.3	Simulation Extractability of Spartan+KZH-Fold	26
8	Implementation and Evaluation	27
8.1	Optimizations	27
8.2	Setup and Registration Costs	29
8.3	Client, Server, and Decryptor Costs	30
8.4	Concrete Comparison with Prior Work	30
A	Additional Preliminaries	38
A.1	Arguments and Commitments	38
A.2	Accumulation and PCD Schemes	39
A.3	Deferred Details of KZH and KZH-Fold	43
B	Zero-Knowledge KZH	43
C	Simulation Extractability	48
C.1	Simulation Extractability of zk-Spartan+zk-KZH	48
C.2	Simulation Extractability of Spartan+KZH-Fold	52

D	Security Proofs of WillowFold	55
D.1	Ciphertext Correlation Intractability	55
D.2	Hybrid Argument for WillowFold	56
E	Concrete Instantiations of Encryption Schemes	58

1 Introduction

Secure aggregation enables one or more servers to compute the sum of clients’ private inputs, while revealing no additional information beyond the final sum. It underpins a wide range of applications, including federated learning [Bon+17; Cho+22], and has recently seen real-world deployments [FL; tec].

Secure aggregation can be realized via a generic secure multi-party computation; however, such an approach typically requires multiple rounds of interaction between the servers and the clients. In most applications, clients are end-user devices (e.g., phones or browsers) that voluntarily provide input to the servers. Hence, lowering the requirements for client participation is a key design goal. The gold standard are *one-shot* protocols like Willow [BC+25] and OPA [KP25], in which clients send a single message to the server(s).

Single-server protocols face a separate challenge, so-called *residual attacks*, in which a malicious server, colluding with a group of clients, tries to learn a (small) subset of contributions that reveals the honest parties’ inputs. To prevent this, a recent line of research [Ma+23; Li+23; BC+25; KP25] relies on a committee of parties that assists the server and ensures that it acts honestly. In more detail, the committee both verifies the server’s work and also decrypts an honest server’s aggregation. Unfortunately, in all prior works, the committee’s total work and communication are linear—or even super-linear—in the number of clients. This is a significant limitation since the committee does not benefit from aggregation and is typically run by volunteers or other altruistic parties. As a concrete example in Willow [BC+25], with $n = 10^7$ number of clients, the committee needs to receive and verify ≈ 15 GB of proofs, taking ≈ 460 hours *in total* to verify them.¹

Strawman Approach to Succinct Verification. A natural way to reduce the committee’s cost is to delegate the verification and aggregation tasks to an untrusted aggregation server using a SNARK (Succinct Non-interactive ARGument of Knowledge). In this setting, the server proves that it has correctly executed the committee’s verification procedure and produced the aggregate ciphertext. This can significantly reduce the committee’s work and communication requirements, keeping them constant or logarithmic in the number of clients. Unfortunately, the verification statement is both asymptotically and concretely large and linear in the number of clients. This renders the use of *monolithic* SNARKs² impractical, as it blows up the server’s memory. Moreover, monolithic approaches suffer from inherent limitations, including the inability to support dynamic client participation and the difficulty of distributing the server’s work across multiple machines.

In this work, we instead use *proof-carrying data* (PCD), also referred to as *recursive* SNARKs. In this paradigm, the statement is decomposed into smaller sub-statements. For each sub-statement, we generate a proof, and then recursively compose these proofs into a single, succinct proof. This approach naturally supports distributed/streaming aggregation and dynamic client participation. On the downside, recursive constructions introduce several additional challenges: (i) The verifier must ensure that there are *no duplicate client contributions* included—a property that is non-trivial to encode in recursive statements; (ii) prior systems such as Willow, rely on non-succinct proofs to verify the aggregation; however, non-succinct proofs are inherently incompatible with recursive proof composition; and (iii) as we show, the security of aggregation systems requires the proof system to satisfy simulation extractability, a strong non-malleability notion. This requirement has largely been overlooked, and most prior efficient recursive proof systems have not been shown to satisfy simulation extractability. In this work, we address these challenges and provide a concretely

¹The verification overhead can be distributed across committee members, therefore, the per-member cost decreases as more members participate in the committee.

²Monolithic SNARKs require the prover to access all wire values of a circuit C and to perform a single global computation over the entire execution trace.

efficient instantiation of a PCD-based secure aggregation that requires minimal client cost and total committee overhead while maintaining reasonable server performance.

Trade-off Between Server Cost and Committee Cost. WillowFold reduces this committee overhead at a non-trivial cost to the server: in our unoptimized benchmarks, the server-side work for $n = 2^{23}$ clients rises by $22\times$. We argue that this asymmetry is precisely what makes the protocol deployable. The server is the party that receives the aggregate and therefore has a direct incentive to provision compute, whereas committee members are typically volunteer or third-party participants who derive no benefit from the aggregation. Volunteer hours are a scarce resource. Moreover, server time and committee time are not measured on comparable hardware. Committee members typically run on client-class devices, while the server runs on elastic datacenter infrastructure: The server in WillowFold can be implemented in a distributed manner using a cluster of machines, whereas committee size is bounded by the honest-majority trust assumption and cannot be elastically scaled.

1.1 Our Contributions

In this paper, we introduce WillowFold, a secure aggregation scheme with lightweight committee cost. WillowFold builds on Willow and reduces the verification cost to be *logarithmic* in the number of clients, which effectively removes the need for a separate resource-intensive verification subcommittee; *any* party—including the decryptors—can efficiently verify the correctness of the aggregation via a succinct, efficiently verifiable proof. Moreover, WillowFold supports *streaming* aggregation and reduces per-client server storage by orders of magnitude compared to Willow. Willow requires the server to *persistently* store all per-client contributions (1.5 KB), whereas in WillowFold the server stores a single ID (4 Bytes) for each client³ and can discard contributions after they are aggregated. This facilitates the distributed execution of the server’s computation. Below, we summarize our contributions.

Tightening Willow’s Security Analysis. We identify a gap in Willow’s security analysis and show that security against *correlated ciphertext* attacks requires the underlying zero-knowledge proofs to satisfy *simulation extractability* (SE). Although the proof system used in the Willow protocol satisfies this property, the formalization has been omitted from the security proof.

Lightweight Secure Aggregation. WillowFold uses PCD to offload the verifier’s work in Willow to the server and reduce the committee’s cost. In particular, the server collects per-client proofs and, using PCD, accumulates them into a succinct proof that certifies the correctness of clients’ contributions and the server’s aggregation. Along the way, we need to overcome multiple challenges:

- We need to mitigate *replay* attacks, in which an adversary may duplicate client contributions. To this end, the PCD statement maintains a binary vector indicating which clients have been aggregated and which have not been. The size of this list is linear in the total number of clients; thus, representing it within the PCD statement exceeds our efficiency goals. To circumvent this, the server commits to one-hot vectors using a homomorphic, constant-size polynomial commitment scheme and distributes these commitments to clients as IDs. Then, in each aggregation session, the server sums the IDs, and finally, at the end, the server produces a succinct proof that the aggregated ID commits to a $\{0, 1\}$ -vector, which implies that no ID was included more than once. This further enables the server to operate in a streaming manner—that is, processing client contributions as they arrive—while using memory that is essentially independent of the number of clients (less than 20 GB in our experiments), modulo a list of previously aggregated IDs.

³For the simulation security proof, the server also stores one public-key encryption ciphertext per client (64 bytes). Dropping this ciphertext does not enable an attack, but it no longer guarantees output correctness (see Remark 1).

- As we discuss later, security against correlated ciphertext attacks requires the PCD scheme to satisfy SE. Recent work [Abd+25] defined SE for general PCDs, where the simulator may be invoked on both leaves and internal nodes. We observe that in the leaf-decoupled setting [Ngu+24], internal nodes are deterministic and carry no witness, so a simulator restricted to leaf queries suffices. We formalize this as leaf-SE, a specialization of [Abd+25] to leaf-decoupled PCDs that lets internal nodes avoid the overhead of zero-knowledge and simulation extractability.
- The client circuit in Willow requires encoding a ring operation in its circuit. A naive implementation of the ring operation would require an $O(d^2)$ -sized circuit for $d = 4096$. We perform a probabilistic, Schwartz-Zippel-based verification of the ring operations, reducing the asymptotic cost to $O(d)$. Furthermore, these operations are non-native to the field used by group-based PCD, causing the circuit to exceed 6 million R1CS constraints. We implement several optimizations that reduce the circuit size to 1.4 million constraints while keeping the witness sparse and low-weight: over 90% of the witness entries are binary, which significantly reduces both prover and client time when using an MSM-based PCD scheme such as Spartan+KZH-Fold.

Concrete Instantiation. We present a concrete instantiation of WillowFold with a variant of Spartan+KZH-Fold [Kad+25], which is a PCD scheme with succinct proof size and verification time. The challenge is that Spartan+KZH-Fold is neither zero-knowledge nor simulation extractable. As part of our contribution, we propose a zero-knowledge variant of Spartan+KZH, which is the SNARK underlying Spartan+KZH-Fold, that uses only a sublinear amount of randomness; we refer to this as zk-Spartan+zk-KZH. We then show that this construction satisfies simulation extractability, which, in turn, yields the variant of Spartan+KZH-Fold built from zk-Spartan+zk-KZH to be leaf-SE.

At the core of zk-Spartan+zk-KZH, we revisit prior work on zero-knowledge sumcheck [CFS17; Xie+19] and introduce a new zero-knowledge sumcheck that uses only a logarithmic amount of randomness and, unlike prior work, can be instantiated with any multilinear polynomial commitment scheme. As an additional contribution, we also revisit the proof of knowledge soundness of Spartan+KZH in the updatable-SRS setting, which we require to prove that zk-Spartan+zk-KZH is SE.

Implementation and Evaluation. We provide an unoptimized implementation, and we benchmark and evaluate WillowFold, demonstrating its practicality for real-world deployments, particularly by eliminating reliance on a committee whose cost scales linearly with the number of clients. For 2^{23} clients with input length $\ell = 10^7$, WillowFold has a proof size of 57 KB and committee computation time of less than one second— $250000\times$ and $2000000\times$ improvements over Willow, respectively. This enables the committee to be run by a small number of lightweight parties, e.g., a subset of the clients themselves. Client computation takes approximately 3.3 seconds, and the server’s computation per client is ≈ 4.3 seconds—representing $3.5\times$ and $22\times$ increases compared to Willow, respectively. However, server storage in WillowFold is only 4 bytes per client, compared to 1.5 KB in Willow—a $380\times$ improvement—thereby enabling more effective streaming aggregation. The client-to-server communication cost remains unchanged relative to Willow, since in both protocols it is dominated by the ciphertext rather than the proof.

1.2 Related Work and Comparison

We compare WillowFold with other committee-based single-server secure aggregation protocols: Flamingo [Ma+23], LERNA [Li+23], OPA [KP25], and Willow [BC+25]. Asymptotic costs are reported in Table 1. As shown, a common limitation of prior work is that the committee cost

scales linearly with the number of clients, which is logarithmic in WillowFold. A detailed discussion appears below.

Flamingo. This scheme relies on *pairwise masking* to realize one-shot secure aggregation. In addition to the committee’s cost, Flamingo has two major limitations. First, it requires a PKI available for all clients. Further, although Flamingo achieves one-shot communication, the subset of clients participating in each session must be fixed and known in advance; thus, it doesn’t support dynamic client participation. On the positive side, Flamingo supports changing the committee members between sessions while re-using the setup via key resharing. Although we do not explore this feature in our work, we believe that similar techniques may also apply to WillowFold.

LERNA. Similar to Willow and WillowFold, LERNA achieves one-shot clients and dynamic client participation using homomorphic encryption. Its main limitation is concrete communication cost—e.g., with 20K clients and input length 50K, LERNA requires 2GB of communication per client and 4.4GB per committee member. LERNA also requires a stateful committee.

OPA. This scheme supports one-shot clients, dynamic participation, and a stateless committee. However, compared to WillowFold, OPA requires a large committee size. In particular, to achieve logarithmic per-member committee cost, OPA requires $O(n)$ committee members, whereas WillowFold uses a constant-size committee. This poses a challenge for large-scale deployments. Therefore, OPA is better suited to settings with multiple iterations and a small number of clients.

Willow. Built on Willow, our scheme inherits many of its properties—one-shot clients, dynamic participation, and a stateful committee. The main difference is the committee cost: it improves from linear in Willow to logarithmic in WillowFold, at the cost of slightly higher server and client overhead (see Section 8.4 for a concrete comparison).

Other Related Work. We do not directly compare our work with single-server secure aggregation schemes that require multiple synchronization rounds between clients [Bon+17; Bel+20]. These protocols do not support scenarios with low client availability and incur high latency. Secure aggregation has also been extensively studied in the multi-server setting [Add+22; CGB]. While this setting can offer better concrete efficiency than single-server protocols, each server incurs communication and computation costs proportional to the number of clients, and the servers must be non-colluding, which introduces logistical constraints. An orthogonal line of work studies input validation in secure aggregation [Bel+23]. We do not consider this aspect; however, we believe such techniques are compatible with WillowFold.

Concurrent Work. TACITA [Mad+25] is a one-shot secure aggregation scheme that supports dynamic participation and further achieves security against “Sybil attacks” where an adversary who controls a large majority of clients, e.g., $n - 1$, can recover honest client inputs. To do this, TACITA employs homomorphic threshold signatures. Their technique, however, incurs a high concrete cost. In particular, while client and committee communication costs are constant, each client’s computation scales linearly with the number of clients. For example, with only 1000 clients, each client’s work takes 32 seconds per aggregation session, while WillowFold has logarithmic client cost and can efficiently handle even millions of clients with about 3 seconds of computation per client. Furthermore, unlike WillowFold, TACITA requires PKI between clients and supports only small input domains. For WillowFold, potential defenses against Sybil attacks include enforcing a high minimum threshold on the number of client contributions or applying differential privacy techniques [BC+25, Appendix I].

Heli [Leh+26] is a concurrent work that builds on techniques from OPA. It proposes a two-server aggregation scheme in which the *aggregator* server performs the bulk of the computation, while the

Protocol	Client	Server	Committee
Flamingo [Ma+23]	$\ell + \log n$	$n(m + \log n + \ell)$	$n \log n$
LERNA [Li+23]	ℓ	$n\ell$	$n + \ell$
OPA [KP25]	$\ell + m$	$n(\ell + m)$	n/M
Willow [BC+25]	ℓ	$n\ell$	$n/M + m$
WillowFold	$\ell + \log n$	$n(\log n + \ell)$	$\log n + m$

Table 1: n is the number of clients, ℓ the client input size, M the number of subcommittees, and m the size of each subcommittee; m is a constant determined by the security parameter and corruption threshold. For OPA and Willow, it additionally scales as $\log M$. In both protocols, there is a trade-off between M and n , since each committee member’s communication and computation costs scale with $O(\frac{n}{M})$. Across all protocols, communication and computation have the same asymptotics, except for Flamingo’s committee, which has communication $O(n \log n)$ and computation $O(n^2)$.

decryptor server assists only with registration and decryption and incurs smaller overhead. Although Heli achieves improved concrete efficiency compared to other multi-server schemes, the decryptor server still has linear overhead in the number of clients: it must register with all clients, and its final decryption computation scales linearly with the number of users who drop out. Finally, as noted above, from a logistical standpoint, it is often more practical to assume that a committee of users does not collude with the central server than to rely on the existence of two non-colluding servers. Relatedly, it is unclear how to realize the decryptor as a lightweight committee.

2 Single-Server Secure Aggregation

In the single-server setting for secure aggregation, a server S aims to compute the sum of n client-held vectors $\vec{x}_i \in \mathbb{F}^\ell$. Communication follows a *star topology* with S at the center; i.e., each client communicates with S over a private, point-to-point channel. As in Willow, our model supports *one-shot* clients with *dynamic participation*, meaning each client sends a single message to S ; the client set doesn’t need to be known or fixed in advance, and clients may join the aggregation session as it progresses. To prevent *replay* attacks, S assigns each client a random identifier ID , and the client attaches to its contribution a proof of possession of ID . This identifier is session-specific and can be issued during the client–server handshake. Furthermore, S interacts with a decryptor role D (which can be instantiated as a decryptor committee) whose public key is known to S and the clients.

Correctness and Output Delivery. Our scheme supports dynamic participation, i.e., clients may drop out without affecting correctness or liveness. Our model is robust to committee-member dropouts when a majority remain available, as in Willow. Output delivery is guaranteed when the server is semi-honest, and committee members are non-adaptive, i.e., they cannot adaptively abort.

Input Validation. We believe that existing techniques from prior work [Bel+23; Cho+22; Lyc+23] can be integrated into WillowFold to ensure well-formed inputs (e.g., low-norm inputs). Input validation is not our focus, and we do not explore it here; instead, we focus on the novel functionality of our construction.

Functionality $\mathcal{F}_{\text{Agg}}$

Parameters: Input domain $\mathbb{F}^\ell$; Number of clients n ; Minimum threshold of number of clients min_t .

Parties: Server S ; Clients $\{C_i\}_{i \in [n]}$ each C_i holding input $\vec{x}_i \in \mathbb{F}^\ell$; $\mathcal{A}$ corrupts $\mathcal{C} \in \{C_i\}_{i \in [n]} \cup \{S\}$.

Functionality:

- $\forall i \in [n]$, $\mathcal{F}_{\text{Agg}}$ receives $\vec{x}_i$ from C_i ; if C_i is corrupted, i.e., $C_i \in \mathcal{C}$, then $\vec{x}_i$ is chosen by $\mathcal{A}$.
- If S is corrupted, $\mathcal{A}$ sends the set of participants $S \subset [n]$ to $\mathcal{F}_{\text{Agg}}$. Otherwise, we set $S := [n]$.
- $\mathcal{A}$ sends $\text{msg} \in \{\text{continue}, \text{abort}\}$ and S to $\mathcal{F}_{\text{Agg}}$.
- If $|S| \geq \text{min}_t$ and $\text{msg} = \text{continue}$, $\mathcal{F}_{\text{Agg}}$ sends $\sum_{i \in S} \vec{x}_i$ to S . Otherwise, $\mathcal{F}_{\text{Agg}}$ returns $\perp$ to S .

Figure 1: The secure aggregation functionality $\mathcal{F}_{\text{Agg}}$.

2.1 Roles and Architecture

We provide an overview of the different roles in our model.

- Clients $\{C_i\}_{i \in [n]}$: Clients provide aggregation inputs, and the goal is to ensure their inputs' privacy. The adversary may actively corrupt up to a threshold number of clients. We do not assume a PKI between clients, but they need to know the decryptor's (the committee's) public key.
- Server S : The server is the entity that receives the final sum but learns no additional information about the clients' inputs. It performs the majority of the work and acts as the coordinator **Coord** for the committee members. We consider both semi-honest and malicious servers in our model.
- Committee $\{D_j\}_{j \in [k]}$: Committee members verify the correctness of aggregation, and assist the server in decryption. We assume that a majority of committee members are honest and do not collude with the server. Moreover, we require their communication and computation costs to be sublinear in both the number of clients n and the input size ℓ . Committee members are stateful because they participate in distributed key generation and must store their shares. We assume PKI among the committee members and that clients can obtain their aggregated public key.

2.2 Secure Aggregation Functionality

The functionality $\mathcal{F}_{\text{Agg}}$ in Figure 1 formalizes our security guarantees. It ensures output delivery whenever the number of contributions exceeds min_t (i.e., can also be set to be 0), and the adversary $\mathcal{A}$ does not abort. By design, $\mathcal{F}_{\text{Agg}}$ permits non-selective aborts: $\mathcal{A}$ must decide to abort before the server learns the output. As in Willow, our protocol can be extended to provide output delivery under selective aborts via an extra round of distributed key generation [BC+25, Section 6.7]).

3 Technical Overview

In this section, we give an overview of our techniques.

3.1 WillowFold Secure Aggregation

We first review the Willow protocol and its verification bottleneck. Then, we present our framework, which overcomes this barrier. Figure 2 illustrates our design and summarizes each role’s task.

Overview of Willow. Willow operates in two phases: offline and online. In the offline phase, the committee and the server (as coordinator) run a distributed key generation that outputs a public key pk for an additively homomorphic encryption scheme AHE and secret shares the decryption key sk among the committee. In the online phase, each C_i submits a homomorphic encryption of its input $\vec{x}_i$, and the committee assists the server in decrypting $\sum_i \vec{x}_i$. In particular, C_i encrypts $\vec{x}_i$ under a symmetric, key and message-homomorphic encryption scheme KAHE and sends

$$a_i = \text{KAHE.Enc}(\vec{x}_i, k_i) \quad \text{and} \quad b_i = \text{AHE.Enc}(k_i, \text{pk}).$$

The committee decrypts $\sum_i b_i$ and sends $k = \sum_i k_i$ to the server, which can then recover $\sum_i \vec{x}_i$.

Correlated Ciphertext Attacks. The scheme described above is insecure when the adversary actively controls clients and observes the server’s view. Suppose the adversary $\mathcal{A}$ corrupts a single client C_{n-1} . After clients $C_1, \dots, C_{n-2}$ submit their contributions, then $\mathcal{A}$ sends $b_{n-1} = -\sum_{i \in [n-2]} b_i$ and $a_{n-1} = -\sum_{i \in [n-2]} a_i$. When client C_n submits its contribution, the committee decrypts

$$\sum_i b_i = \text{AHE.Enc}\left(\sum_i k_i, \text{pk}\right) = \text{AHE.Enc}(k_n, \text{pk}),$$

which allows S to recover $\sum_i \vec{x}_i = \vec{x}_n$, the input of C_n . To resolve this, Willow requires each C_i to attach π_i , a *zero-knowledge proof of knowledge* of k_i , which ensures that it is sampled independently.

This observation, however, must be formalized, which is omitted from Willow’s security proof. To fill this gap, we define security against ciphertext correlation attacks and show that if the proofs π_i satisfy simulation extractability (SE)—a strong non-malleability notion—the scheme is secure against such attacks. The intuition is that an adversary can produce a proof only for a ciphertext whose encryption key and randomness it knows, and this must hold even *after* the adversary observes honest proofs for other ciphertexts (further details in Section 6 and Appendix D.1).

Lightweight Secure Aggregation via PCD. In Willow, there is a verifier committee that verifies each client’s proof π_i individually and performs the ciphertext summation. This requires a linear cost in the number of clients. In WillowFold, the server instead directly computes the combined ciphertext and generates a succinct proof that the decryptor committee can efficiently verify. To realize this, we use proof-carrying data (PCD). A PCD scheme enables parties in a distributed computation to output proofs that they have correctly performed their share of the work; these proofs can be recursively composed and incrementally accumulated into a single, succinct proof. In WillowFold, clients send the server their encryption, along with proofs of contribution as in Willow. The server collects all per-client proofs and, via PCD, accumulates them into a single proof. The same proof also shows that the ciphertexts are correctly summed. The server sends the aggregated ciphertext and the final proof to the committee—whereas Willow forwards all ciphertexts and per-client proofs. The final proof certifies all per-client proofs and the server’s work.

This approach also enables streaming aggregation. It reduces the server’s storage. Once a contribution arrives, the server immediately accumulates it into the running proof and ciphertext and can discard the raw input without waiting for other contributions. Even with PCD, streaming requires careful treatment of client IDs to prevent replay attacks. Our solution requires the server to only track which clients have participated by storing a small per-client ID. This contrasts with Willow, which requires storing all client proofs and ciphertexts until the end of the protocol.

To further enable parallelism in the server’s computation and improve its performance, we use tree-based PCDs in a leaf-decoupled mode [Ngu+24], where per-client proofs and ciphertexts reside at the leaves, while each internal node verifies and aggregates its children. The leaf-decoupling technique reduces the circuit size at both the leaf and internal levels by eliminating unnecessary computations. The root node then recursively proves the correctness of the entire tree. In contrast to straight-line accumulation—also known as IVC [Val08]—tree-based PCD supports parallel/distributed proving.

ID Registration and Replay Attacks. A challenge is that an adversarial server may duplicate client contributions and run *replay attacks*. To avoid this, Willow assigns each client a unique random ID, which the client includes in its contribution. Then, the verifier committee checks that no duplicate IDs appear in the contributions. However, in WillowFold, the issue is more involved; we cannot send IDs to the committee, since this incurs a linear cost for committee members. Additionally, enforcing the uniqueness of client IDs directly within the PCD circuit would require a circuit whose size scales linearly with the number of clients, undermining our efficiency goals.

Instead, we apply the following technique. Let $\text{com}(\cdot)$ be a homomorphic vector (equivalently, polynomial) commitment such that $\text{com}(\vec{a} + \vec{b}) = \text{com}(\vec{a}) \cdot \text{com}(\vec{b})$. The server assigns each C_i a unique ID $g_i = \text{com}(\vec{e}_i)$, where $\vec{e}_i$ encodes a one-hot vector of length n (total number of clients), i.e., its i th entry is 1, and others are 0. Here we assume $\text{com}(\cdot)$ is deterministic; hence, the IDs cannot be randomized. The server maintains a participation vector pv initialized as $\text{pv} := 0$ with commitment $\text{com}(\text{pv}) = 1$. When C_i submits its contribution, the server updates $\text{pv} \leftarrow \text{pv} + \vec{e}_i$ and $\text{com}(\text{pv}) \leftarrow \text{com}(\text{pv}) \cdot g_i$. This vector is stored efficiently by simply storing the set of client IDs. The PCD circuit verifies that these updates are correct, where adding two commitments is a constant-size operation, e.g., a single group addition. Finally, the server proves to the committee that $\text{com}(\text{pv})$ commits to a $\{0, 1\}$ -valued vector, ensuring there is no duplication. Such a proof can be generated via a *zerocheck*, with proof size and verification time $O(\log n)$.

What remains is to certify that IDs are well-formed and that each ID indeed commits to a one-hot vector. To ensure this, we run a one-time setup to generate a public Merkle tree $\mathcal{T}$ whose leaves contain all valid IDs; both the IDs and the tree are publicly verifiable. The Merkle root serves as a commitment to the set of IDs. When the server assigns a client an ID g_i , it also provides the corresponding Merkle path. The client then includes g_i and its Merkle path in its contribution, and the PCD circuit checks its consistency with the Merkle root. Note that the same tree $\mathcal{T}$ can be reused across many aggregation sessions, and IDs need to be unique within a given session.

Extraction from PCD Proofs. For simulation security, we require the simulator to extract the IDs of honest clients who participated in the aggregation—this enables the simulator to learn the sum of the honest clients’ inputs and to reprogram the random oracle accordingly. In Willow, such extraction is straightforward: the server forwards all per-client proofs, and the simulator runs the extractor of the proof system to obtain the IDs. In WillowFold, the server forwards only the PCD root. However, the definition of leaf-SE for PCDs (see Definition C.7) guarantees an efficient extractor that, given the root, can recover the entire PCD transcript, including the leaf nodes and their witnesses. Hence, a simulation-based security proof analogous to Willow also applies here.

Distributed and Parallel Aggregation. An advantage of WillowFold is that the server’s work is amenable to a high degree of parallelism and distributed execution across many machines. The server workflow has three phases: (i) a one-time setup to build $\mathcal{T}$; (ii) PCD computation; and (iii) a zerocheck proving ID uniqueness. Computing a Merkle tree and performing PCD aggregation can be naturally distributed. However, the *zerocheck* in principle is difficult to distribute. Nevertheless, in practice, it can be executed effectively by a single server. The support for parallel and distributed computation comes from our use of PCD. If one instead applies a monolithic SNARK to Willow’s

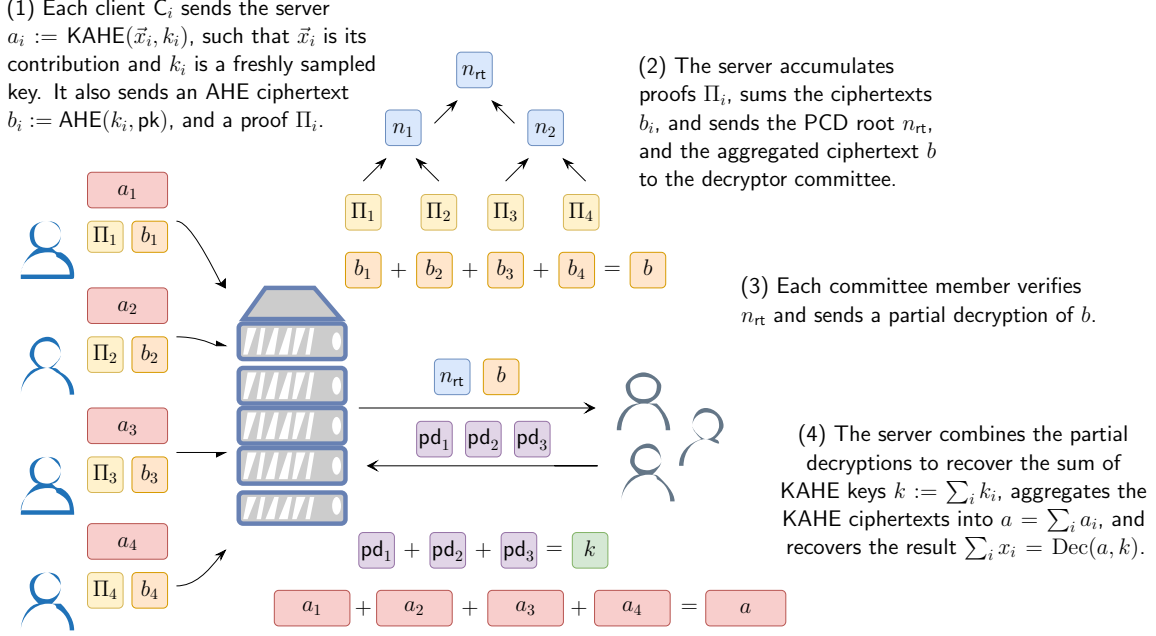


Figure 2: An overview of WillowFold. Each client sends an encryption of its contribution and attaches a proof. The server accumulates client proofs and sends the final proof to the committee. The committee checks the proof and sends the partial decryption of *the sum of the symmetric keys* $k = \sum_i k_i$, enabling the server to recover k and, in turn, the sum of client contributions $\sum_i \vec{x}_i$.

verification statement, it is unclear how to parallelize or distribute the computation, since the non-duplication check does not naturally decompose into independent chunks.

3.2 Requirements of the PCD Scheme

In this subsection, we outline the properties required of the underlying PCD scheme, justify our choice of leaf-decoupled PCD, and provide an overview of our notion of SE for leaf-decoupled PCDs.

PCD Requirements. For the security of WillowFold, we require the PCD scheme to satisfy two properties: (i) support accumulation over zero-knowledge proofs—that is, in WillowFold, the leaves of the PCD tree are per-client proofs, which must hide the keys used to encrypt client inputs; and (ii) be leaf-SE to ensure security against correlated ciphertext attacks. For efficiency, we also require the PCD to be *sublinear* [Kad+25], with proof size and verification time sublinear in the size of the statement. This ensures that clients’ and committee members’ communication, as well as the server’s computational overhead for verifying client proofs, remains concretely small.

Leaf-Decoupled PCD. As mentioned above, we use PCDs in a leaf-decoupled mode in which the PCD directed acyclic graph (DAG) forms a complete tree: leaves perform witness-dependent computations, while internal nodes carry no witness and only accumulate their children’s outputs. This design improves efficiency by reducing circuit size at both the leaf and internal levels [Ngu+24]. This architecture is particularly well-suited to our setting: clients generate the leaf proofs, and a server aggregates them into a single proof, where aggregation involves no secret witness.

PCD Simulation Extractability. SE for a zero-knowledge proof system is typically defined by giving the adversary access to an oracle that produces simulated proofs for arbitrary queried

statements; SE then guarantees that the adversary cannot forge a satisfiable proof—that was not queried to the simulator—for any statement without *knowing* the corresponding witness; in particular, there is an efficient extractor algorithm that can extract the witness from the adversary.

Defining SE for PCDs, however, requires additional care. An adversary may query the simulator for a proof π and then run the proving algorithm to extend it into a fresh PCD proof π^* that has not been queried, yet π^* may not be extractable since it builds on the simulated proof π . Such attacks do not arise in non-recursive proofs. A concurrent work [Abd+25] introduces a notion of SE for PCDs, in which the extractor is required to recover the witnesses only for non-simulated nodes.

In our leaf-decoupled PCD architecture, leaf nodes must be zero-knowledge and simulation extractable. In contrast, internal nodes are deterministic and carry no witness, and thus need not be zero-knowledge, and consequently need not be simulation extractable, since there may be no simulator for them. Motivated by this observation, we relax the notion of SE for PCDs from [Abd+25] by requiring the simulator to answer only leaf queries. We then show that a leaf-decoupled PCD satisfies this notion when the leaf nodes are SE and the internal nodes are knowledge sound.

3.3 Instantiation with Spartan+KZH-Fold

We instantiate WillowFold with Spartan+KZH-Fold [Kad+25], a sublinear PCD scheme.

KZH-Fold. Recent work [Kad+25] built KZH, a polynomial commitment scheme with succinct openings, and further constructed KZH-Fold, a *sublinear accumulation scheme* for the KZH verifier (see Appendix A.3). Using a standard compilation [Chi+20], they instantiated Spartan [Set20] with KZH to obtain a SNARK, which we refer to as Spartan+KZH. They then applied KZH-Fold to obtain a PCD scheme, called Spartan+KZH-Fold. Due to the sublinearity of KZH and KZH-Fold, this PCD scheme has sublinear proof size and verification time. Despite this sublinearity, Spartan+KZH-Fold further achieves prover time comparable to state-of-the-art PCD schemes such as Nova [KST22]. These features motivate our choice of Spartan+KZH-Fold as the PCD backend for WillowFold.

However, Spartan+KZH-Fold is neither zero-knowledge nor known to be simulation extractable. To address this gap, we first present a zero-knowledge variant of Spartan+KZH and then show that it satisfies SE, which in turn implies that Spartan+KZH-Fold is a leaf-SE PCD scheme.

To realize zero knowledge for Spartan+KZH, we note that its two primary components are: Spartan and KZH polynomial commitments. Prior work [Haf+25] shows how to achieve zero-knowledge for KZH commitments. Fortunately, this transformation introduces only *minimal* changes to the KZH verification algorithm, and the original accumulation scheme (i.e., KZH-Fold) can still be used—with minimal modifications—to accumulate zk-KZH. The Spartan polynomial IOP can also be made zero-knowledge via known zero-knowledge sumcheck schemes [Xie+19] with only logarithmic randomness. Putting together these yields a zero-knowledge variant of Spartan+KZH, which we call zk-Spartan+zk-KZH. However, there is a mismatch: techniques for zero-knowledge sumcheck require commitments to multivariate polynomials with individual degrees > 1 . KZH supports only multilinear polynomials. To address this issue, we propose a method for efficiently committing to higher-degree sparse polynomials using multilinear polynomial commitments, while keeping the zero-knowledge overhead logarithmic. Further details appear in Section 7.2.

We show that zk-Spartan+zk-KZH satisfies SE. Prior work [Gan+22] identifies sufficient conditions for SE in proof systems with an updatable setup: (i) *knowledge soundness*, (ii) *unique response*, and (iii) *trapdoor-less zero-knowledge*. We prove that zk-Spartan+zk-KZH satisfies all three properties and achieves SE. As mentioned before, KZH-Fold—with minor modifications—can be applied to zk-Spartan+zk-KZH to obtain a PCD scheme. Since KZH-Fold satisfies knowledge

soundness [Kad+25] and zk-Spartan+zk-KZH satisfies SE, the leaf-decoupled PCD scheme built from these satisfies leaf-SE.

We note that WillowFold can be generically instantiated with any leaf-SE PCD scheme. As discussed above, sublinearity of PCD is crucial for reducing both client-server communication and the server's computational overhead. Thus, this observation suggests future work on identifying alternative PCD schemes beyond Spartan+KZH-Fold, potentially improving efficiency.

Arithmetization of Ring Operations. Similar to Willow, WillowFold uses an AHE scheme based on RLWE. Spartan+KZH-Fold (which is based on R1CS) does not natively support ring operations, which would incur high arithmetization overhead. More concretely, the client's circuit includes a product of two ring elements, e.g., $\mathbf{w} = \mathbf{u} \cdot \mathbf{v}$. Each ring element is a polynomial of degree d ; thus, this product relation adds $O(d^2)$ constraints. To avoid this, we verify each product via a probabilistic check: evaluate the underlying polynomials at a random challenge r and show that $w(r) = u(r) \cdot v(r)$, which requires only $O(d)$ constraints. A subtle issue is efficiently deriving the challenge r ; to do so, we use the *commit-and-prove* paradigm [CFQ19], as described in Section 8.1.

4 Preliminaries

We write $[n] = \{1, 2, \dots, n\}$. For a vector $\vec{v}$, let v_i denote its i th entry. We use $\mathbb{F}$ to denote a finite field and $\mathbb{G}$ to denote a multiplicative group with scalars in $\mathbb{F}$. We use $e(\cdot, \cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ to denote a bilinear pairing map. $\mathbb{F}_\mu^{\leq d}[\vec{X}]$ denotes the set of multivariate polynomials defined over μ variables, in which each term has a degree $\leq d$. We let λ be the security parameter and $\text{negl}(\lambda)$ a negligible function in λ . An algorithm is *efficient* if its running time is a polynomial in λ .

We define $R_q = \mathbb{F}[X]/(X^N + 1)$, the ring of all polynomials over the finite field $\mathbb{F}$ where $|\mathbb{F}| = q$, where addition and multiplication are performed modulo the polynomial $X^N + 1$. Equivalently, each element of R_q is uniquely represented by a polynomial of degree $< N$ with coefficients in $\mathbb{F}$.

Multilinear Extensions. A *multilinear* polynomial is a multivariate polynomial with individual degree at most 1. A multilinear polynomial $p(\vec{X})$ over ℓ variables is uniquely specified by its $N = 2^\ell$ evaluations on the Boolean hypercube $\{0, 1\}^\ell$. For $\vec{w} \in \{0, 1\}^\ell$, we let $\text{eq}(\vec{w}, \vec{X})$ denote the multilinear Lagrange basis that equals 1 at $\vec{X} = \vec{w}$ and 0 elsewhere in $\{0, 1\}^\ell$, in other words, $\text{eq}(\vec{w}, \vec{X}) = \prod_{i=1}^\ell (X_i w_i + (1 - X_i)(1 - w_i))$. The *multilinear extension* of a vector $\vec{v} \in \mathbb{F}^{2^\ell}$ is a unique multilinear polynomial $\tilde{v}(\vec{X})$ s.t. $\tilde{v}(\vec{x}) = v_{\vec{x}}$ for all $\vec{x} \in \{0, 1\}^\ell$, i.e., $\tilde{v}(\vec{X}) = \sum_{\vec{w} \in \{0, 1\}^\ell} \text{eq}(\vec{w}, \vec{X}) \cdot v_{\vec{w}}$.

4.1 Argument Systems

A non-interactive argument of knowledge (NARK) for a relation $\mathcal{R}$ enables a prover to convince a verifier that it knows a witness w such that $(x, w) \in \mathcal{R}$. Formally, it has the following syntax:

- $\text{Setup}(1^\lambda, R) \rightarrow \text{pp}$: given λ and relation R , outputs public parameters pp .
- $\text{Prove}(\text{pp}, x, w) \rightarrow \pi$: given pp , an instance x , and a witness w , outputs a proof π .
- $\text{Verify}(\text{pp}, x, \pi) \rightarrow \{0, 1\}$: given pp , an instance x , and a proof π , outputs either accept or reject.

A NARK is *complete* if, under an honest setup, whenever $(x, w) \in \mathcal{R}$, the verifier accepts the proof output by an honest prover. It is *knowledge sound* if there exists an efficient extractor that can extract a witness from any malicious prover P^* that outputs an accepting proof. It is *zero knowledge* if there exists an efficient simulator Sim that, given only the statement x , outputs a proof π that is indistinguishable from the proof produced by Setup and Prove . Finally, *simulation extractability*

strengthens knowledge soundness by requiring extraction to hold even when a malicious prover P^* is given oracle access to Sim . Formal definitions of properties appear in Appendix A.1.

Sumcheck. The *sumcheck protocol* [Lun+90] enables a prover to convince a verifier of claims of the form $H = \sum_{\vec{x} \in \{0,1\}^\ell} p(\vec{x})$, where $p(\vec{X})$ is a multivariate polynomial with individual degree d . This protocol reduces such a claim to checking an evaluation of p at a random challenge point $\vec{r}$. It achieves proof size $O(d\ell)$, prover time $O(2^\ell)$, and verifier time $O(d\ell)$.

Zerocheck. The *zerocheck protocol* [Che+23] verifies that an ℓ -variate polynomial $p(\vec{X})$ of individual degree d evaluates to zero on all points of $\{0,1\}^\ell$. One possible instantiation runs a sumcheck on $0 = \sum_{\vec{x} \in \{0,1\}^\ell} p(\vec{x}) \cdot \text{eq}(\vec{r}, \vec{x})$, for a random challenge $\vec{r} \in \mathbb{F}^\ell$, yielding the same cost as sumcheck.

Accumulation Scheme. An accumulation scheme [BC23] is a core primitive in recursive proofs. At a high level, it enables a prover and a verifier to reduce the satisfiability of two or more NP statements into a single statement of the same size via a lightweight interaction between the two parties. For a formal definition, we refer the reader to Appendix A.2.

Proof-Carrying Data. Proof-carrying data (PCD) [CT10] is a recursive argument of knowledge that enables incremental verification of distributed computations. In particular, suppose we aim to verifiably compute $F^n(\cdot)$, i.e., the n -fold iterated application of a function F . This is known as *incremental verifiable computation (IVC)* [Val08]. A generic approach applies a succinct argument at each iteration i to a circuit C that (i) computes the i th step of F and (ii) verifies a proof π_{i-1} of correctness for the previous step; thus, each proof recursively attests to all prior iterations. Recent works [BGH19; Bün+21a; KST22] study IVC via *accumulation*, where—rather than verifying proofs inside the circuit—we accumulate the verification statements and perform a single verification at the end.

PCD generalizes IVC to distributed computations over directed acyclic graphs (DAGs). Each node may receive inputs and proofs from multiple predecessors; it verifies them, executes its local computation, and outputs the result along with a proof. A PCD scheme is *sublinear* [Kad+25] if, at each node, the proof size and verification time are sublinear in the size of the computation performed at that node. Sublinearity is essential because PCD proofs typically dominate communication between parties, and each proof must be verified by subsequent nodes. Without sublinearity, PCD incurs both high communication cost and computation overhead in distributed environments.

Leaf-decoupled PCD [Ngu+24] is a specific case in which the actual computation is run at the leaf nodes of a tree, and each internal node only verifies its children. The root node recursively verifies the entire tree. More formally, the proving algorithm in a leaf-decoupled PCD scheme is decoupled into two subalgorithms P_{leaf} and P_{internal} such that P_{leaf} outputs leaf proofs and P_{internal} outputs internal proofs. Additional background and formal definition are provided in Appendix A.2.

4.2 Commitment Schemes

Merkle Tree. A Merkle tree $\mathcal{T}$ is a data structure that commits to a vector $\vec{v} \in \mathbb{F}^n$ as follows. The leaves store the (hash of) elements of $\vec{v}$, each internal node stores the hash of the concatenation of its children, and the root hash serves as a commitment to $\vec{v}$. To open the commitment at position i , the prover returns leaf v_i together with *sibling nodes* in the path from the leaf v_i to the root.

Polynomial Commitments. A polynomial commitment scheme (PCS) is a primitive that allows a prover to succinctly commit to a polynomial $p \in \mathbb{F}_\mu^{<d}[\vec{X}]$ and later prove the correctness of its evaluation at chosen points, without revealing the polynomial itself. Upon receiving a request from the verifier to open the commitment at a point $\vec{x}$, the prover returns the claimed value $y = p(\vec{x})$.

together with a proof π attesting to its correctness. Using π , the verifier can check that y is indeed the evaluation of the committed polynomial at $\vec{x}$. Formally, a PCS follows the syntax below:

- $\text{Setup}(1^\lambda, k) \rightarrow \text{pp}$: given λ and degree bound k , outputs public parameters pp .
- $\text{Com}(\text{pp}, p(\vec{X})) \rightarrow c$: given pp and a polynomial $p(\vec{X})$, outputs a commitment c .
- $\text{Open}(\text{pp}, p(\vec{X}), \vec{x}) \rightarrow (y, \pi)$: given pp , $p(\vec{X})$, and point $\vec{x}$, outputs $y = p(x)$ and a proof π .
- $\text{Verify}(\text{pp}, c, \vec{x}, y, \pi) \rightarrow \{0, 1\}$: given pp , c , $\vec{x}$, y , and π , accepts or rejects.

A PCS is *hiding* if the commitment is hiding and the opening is a zero-knowledge argument. It is *extractable* if the commitment is binding and the proof is a knowledge sound argument. Formal definitions appear in Appendix A.1. A PCS is *additively homomorphic* if, for commitments c_1 and c_2 to polynomials f_1 and f_2 , respectively, $c_1 + c_2$ is a valid commitment to the polynomial $f_1 + f_2$.

4.3 Homomorphic Encryption Schemes

Similar to Willow, our construction employs two different homomorphic encryption schemes: (i) KAHE, a symmetric-key, key-and-message additive homomorphic encryption scheme, and (ii) a public-key, threshold homomorphic encryption scheme AHE. We discuss the syntax of each below.

KAHE. For our KAHE scheme, we require that, for any pair of messages m_1, m_2 and keys k_1, k_2 , given ciphertexts ct_1 and ct_2 that encrypt m_1 and m_2 under keys k_1 and k_2 , respectively, $\text{ct}_1 + \text{ct}_2$ is an encryption of $m_1 + m_2$ under the key $k_1 + k_2$. We further require security to hold even if the adversary learns the sum of the keys (see leakage-resilient security in Appendix D). Such a KAHE scheme can be instantiated under the RLWE assumption [BC+25]. The syntax of KAHE is as follows:

- $\text{Setup}(1^\lambda) \rightarrow \text{pp}$: given λ , outputs the public parameters pp .
- $\text{KeyGen}(\text{pp}) \rightarrow k$: given pp , outputs a random secret key k .
- $\text{Enc}(\text{pp}, m, k) \rightarrow \text{ct}$: given pp , a message m , and a key k , outputs a randomized ciphertext ct .
- $\text{Dec}(\text{pp}, \text{ct}, k)$: given pp , a ciphertext ct , and the secret key k , outputs the plaintext m .

AHE. The AHE scheme is required to be additive threshold homomorphic: given ciphertexts ct_1 and ct_2 that encrypt messages m_1 and m_2 under the same public key pk , $\text{ct}_1 + \text{ct}_2$ is a valid ciphertext encrypting $m_1 + m_2$ under pk . To prevent *rogue-key attacks*, each party, when publishing its partial public key, must also provide a proof of knowledge of that partial secret key. In our setting, to further protect against correlated ciphertext attacks, every encrypting party additionally supplies a *proof of plaintext knowledge* corresponding to the ciphertext it produces. Our concrete instantiation of AHE (which follows Willow) is realized under the RLWE assumption [BC+25]. The syntax is as follows:

- $\text{Setup}(1^\lambda) \rightarrow \text{pp}$: given λ , outputs public parameters pp .
- $\text{VerifiableKeyGen}(\text{pp}) \rightarrow (\text{pk}_i, \text{sk}_i, \pi_i)$: given pp , each party outputs a partial public key pk_i , a partial secret key sk_i and a proof of knowledge of the secret key π_i .
- $\text{VerifyKeyGen}(\text{pp}, \text{pk}_i, \pi_i) \rightarrow \{0, 1\}$: given pp , pk_i and π_i , accepts or rejects.
- $\text{AggregateKey}(\text{pp}, \{\text{pk}_i\}_i) \rightarrow \text{pk}$: given pp and $\{\text{pk}_i\}_i$, outputs an aggregated public key pk .

- $\text{VerifiableEnc}(\text{pp}, \text{pk}, m) \rightarrow (\text{ct}, \pi)$: given pp , pk , a message m , outputs a randomized ciphertext ct and a proof of knowledge of the plaintext π .
- $\text{VerifyEnc}(\text{pp}, \text{pk}, \text{ct}, \pi) \rightarrow \{0, 1\}$: given pp , pk , ct , and π , accepts or rejects.
- $\text{ParDec}(\text{pp}, \text{ct}, \text{sk}_i) \rightarrow \text{pd}_i$: given pp , ct , and sk_i , outputs partial decryption pd_i .
- $\text{Recover}(\text{pp}, \text{ct}, \{\text{pd}_i\}_i) \rightarrow m$: given pp , ct , and $\{\text{pd}_i\}_i$, outputs plaintext m .

Our concrete instantiations of these two encryption schemes follow Willow and are presented in Appendix E. One minor detail concerns the choice of the underlying proofs of knowledge. For the AHE encryption scheme, part of the encryption procedure is executed inside the PCD circuit, which serves as a proof of knowledge of the underlying message. For AHE key generation—run by the committee members—the associated proof of knowledge can be chosen arbitrarily, e.g., the Bulletproofs-based instantiation proposed in Willow.

5 Secure Aggregation with a Lightweight Committee

We introduce WillowFold by outlining its setup and the protocols carried out by each role.

5.1 Setup and Registration

Setup. The setup phase outputs: (i) public parameters of PCS, PCD, KAHE, and AHE and (ii) a Merkle tree $\mathcal{T}$ built over the client IDs. During this phase, the committee, with the assistance of the server, also execute the distributed key generation (DKG) protocol, which distributes shares of the AHE secret key among the committee members. We describe DKG in more detail in Section 5.2. The setup protocol is illustrated in Figure 3. If the setup requires no secret randomness, it can be executed directly by the server (and by the committee members when running the DKG). Otherwise, if it requires a trusted setup—such as our instantiation of PCD with Spartan+KZH-Fold—then the components that require secret randomness are generated by a trusted entity. In practice, such a trusted setup is typically implemented via an MPC ceremony involving multiple parties [WCB25].

Registration. Whenever a client wishes to contribute, it first performs a handshake with the server to obtain an identifier g_i together with a Merkle proof of membership π_i attesting that g_i belongs to $\mathcal{T}$. Importantly, the server does not need to maintain any mapping between clients and their assigned g_i . The particular assignment is immaterial, provided that the client can demonstrate the authenticity of g_i via the Merkle proof. The server only keeps track of which generators have already been issued. The registration protocol is also illustrated in Figure 3.

5.2 Client, Server, and Decryptor Roles

Below, we outline the protocols executed by each role in WillowFold.

Client Role. Each client C_i samples a seed s_i and masks its input $\vec{x}_i$ with $\text{PRG.Expand}(s_i)$, where PRG is a pseudorandom generator built from a random oracle. This step is necessary since the secure aggregation functionality (Figure 1) is impossible to realize in the plain model [BC+25, Theorem 1], and we rely on the random oracle to circumvent this barrier. C_i sends to the server an encryption of the seed s_i under the committee’s auxiliary public key pk_{aux} . It also sends an encryption of its masked input $\vec{y}_i = \vec{x}_i + \text{PRG.Expand}(s_i)$ under KAHE, together with an AHE encryption of the KAHE key k_i under the committee’s main public key pk . Finally, the client provides a PCD leaf

Setup($1^\lambda, N$) $\rightarrow$ **pp** : Given λ and a bound on the number of users N , outputs public parameters **pp**:

1. Run **PCS.setup**($1^\lambda, N$) $\rightarrow$ **pp_{PCS}**. Then, define the set of client IDs $(g_0, g_1, \dots, g_{N-1})$ such that $g_i = \text{PCS.Com}(\text{pp}_{\text{PCS}}, \tilde{e}_i)$ where $\tilde{e}_i$ is the multilinear extension of the i th unit vector of size N .
2. Sample H from a hash function family $\mathcal{H}$ and compute a Merkle tree $\mathcal{T}$ over $(g_0, g_1, \dots, g_{N-1})$ using H . Let **rt** be the Merkle root, which will be used for membership verification.
3. Run **PCD.Setup**(1^λ) $\rightarrow$ **pp_{PCD}** to generate the public parameters of PCD. This setup may be circuit-specific and take the step circuit as input or be universal as in Spartan+KZH-Fold.
4. Run **KAHE.Setup**(1^λ) $\rightarrow$ **pp_{KAHE}**, **AHE.Setup**(1^λ) $\rightarrow$ **pp_{AHE}**, **E.Setup**(1^λ) $\rightarrow$ **pp_E** to generate the public parameters of encryption schemes (see Appendix E for concrete instantiations).
5. Run DKG as described in Figure 6 and forward the resulting keys **pk** and **pk_{aux}** to the server.
6. Output the public parameters **pp** := (**pp_{PCS}**, **pp_{PCD}**, **rt**, **pp_{KAHE}**, **pp_{AHE}**, **pp_E**, **pk**, **pk_{aux}**).

Register() $\rightarrow$ (**rt**, g_i , π_i) : Outputs the root **rt**, an ID g_i , and a proof π_i :

1. Retrieve **rt** corresponding to the Merkle tree $\mathcal{T}$ over the set of IDs $(g_0, \dots, g_{N-1})$.
2. Select an unassigned ID g_i from the set of IDs, and mark it as assigned.
3. Let π_i denote the Merkle authentication path from the leaf g_i to the root **rt**.
4. Output (**rt**, g_i , π_i).

Figure 3: Setup and client registration subroutines.

proof that the AHE encryption of k_i is a valid ciphertext, and it owns a correct ID by including the ID and its Merkle tree membership proof in the statement. The protocol is shown in Figure 4. A subtle aspect of the client’s statement to be proven within the PCD circuit is that, due to the lattice-based construction of AHE, it entails a single ring multiplication and multiple range proofs. In Sections 3.3 and 8.1, we present optimizations that enable efficient proofs of these operations.

Remark 1. We emphasize that the client’s use of the PRG is required only for simulation-based security with respect to the functionality in Figure 1. Willow introduces an alternative functionality called *Manipulable Secure Aggregation* [BC+25, Figure 9], which satisfies a weaker security notion where an adversary *corrupting the server* may choose the final output of the aggregation, thus giving up correctness in that case, while preserving input privacy. On the positive side, a simpler protocol variant without PRG masking suffices in that case. WillowFold naturally extends to this variant.

Server Role. The server plays a central role in our protocol and is responsible for the following tasks: (i) It assists in the setup phase by *possibly* generating the system’s public parameters⁴ and supports the decryptors in executing the DKG; and (ii) it receives the clients’ contributions and aggregates them into a single ciphertext and PCD proof. The server then generates a proof of uniqueness of IDs, ensuring that no contribution is duplicated, and sends this proof to the decryptors. Upon receiving the partial decryptions, the server can recover the sum of client inputs. The protocol is formally illustrated in Figure 5.

⁴As explained in Section 8.2, the server executes all setup components that do not require any secret randomness, such as generating the Merkle tree. Other setup procedures (e.g., KZH setup) require a trusted entity.

Each client C_i holds an input $\vec{x}_i \in \mathbb{F}^\ell$ and proceeds as follows:

1. Receive the ID and the public keys:

- (a) Receive $(\mathcal{D}, \Sigma, \mathbf{pk}, \mathbf{pk}_{\text{aux}})$ from the server.
- (b) Check that $|\Sigma| \geq t$ and signatures in Σ are valid signatures on $(\mathbf{pk}, \mathbf{pk}_{\text{aux}})$; otherwise abort.
- (c) Call functionality **Register** (see Figure 5.1) and receive the ID tuple $(\mathbf{rt}, g_i, \pi_i)$ from the server.

2. Mask the input and compute ciphertexts:

- (a) Run $\text{KAHE.KeyGen}(\text{pp}_{\text{KAHE}}) \rightarrow k_i$ and $\text{PRG.KeyGen}(1^\lambda) \rightarrow s_i$.
- (b) Compute the masked input $\vec{y}_i := \text{PRG.Expand}(s_i, \ell) + \vec{x}_i$.
- (c) Run $\text{KAHE.Enc}(\text{pp}_{\text{KAHE}}, \vec{y}_i, k_i) \rightarrow \text{ct}_i^{\text{masked}}$.
- (d) Run $\text{AHE.Enc}(\text{pp}_{\text{AHE}}, \mathbf{pk}, k_i) \rightarrow \text{ct}_i^{\text{key}} := (\text{ct}_0, \text{ct}_1)$.
- (e) Run $\text{E.Enc}(\text{pp}_{\text{E}}, \mathbf{pk}_{\text{aux}}, s_i) \rightarrow \text{ct}_i^{\text{seed}}$.

3. Generate a proof of correct contribution:

- (a) Construct a zero-knowledge PCD base proof Π_i^{PCD} for a relation $\mathcal{R}_i^{\text{mr}} \wedge \mathcal{R}_i^{\text{Enc}}$, where $\mathcal{R}_i^{\text{mr}}$ and $\mathcal{R}_i^{\text{Enc}}$ are defined as follows. Let $\mathcal{X}_i^{\text{PCD}} := (\mathbf{rt}, g_i, \mathbf{u}, \text{ct}_1)$ be the instance for this relation.
 - Membership proof: The client includes its ID g_i as input to the circuit and proves that g_i is a valid element in the server's Merkle tree $\mathcal{T}$ with root $\mathbf{rt}$, i.e.,

$$\mathcal{R}_{\text{mr}} := \{(x := (g_i, \text{root}), w := \pi_i) : \text{MR.Verify}(\text{root}, g_i, \pi_i) = 1\}.$$

- Encryption with constraints: The client encrypts k_i using AHE. Within the circuit, the client proves that the ciphertext is valid, i.e. the key and the ciphertext randomness satisfy the following constraints. See Appendix E for details of the AHE scheme.

$$\mathcal{R}_{\text{Enc}} = \{(x := (\mathbf{u}, \text{ct}_1); w := (\mathbf{v}, e^1)) \mid \text{ct}_1 = \mathbf{u} \cdot \mathbf{v} + e^1 \bmod q_2, \|\mathbf{v}\| \leq B_s, \|e^1\| \leq B_e\}.$$

4. Send ciphertexts and proofs:

- (a) C_i sends $((\text{ct}_i^{\text{masked}}, \text{ct}_i^{\text{key}}, \text{ct}_i^{\text{seed}}), (\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}}))$ to the server.

Figure 4: The client role protocol.

The server S proceeds as follows during setup and aggregation.

1. **Get the public parameters and the committee's public keys:**

- (a) Run setup (see Figure 3). Let $\mathcal{K} = (\mathcal{D}, \Sigma, \text{pk}, \text{pk}_{\text{aux}})$, where $\mathcal{D}$ denotes the set of decryptors participated in the DKG with $|\mathcal{D}| > t$, and Σ denotes the set of their signatures on $(\text{pk}, \text{pk}_{\text{aux}})$.
- (b) Set $\text{encSum} \leftarrow 0$, $\text{encKey} \leftarrow 0$, $\text{encSeeds} \leftarrow \text{emptylist}$, $\text{proofs} \leftarrow \text{emptylist}$, and $\text{IDs} \leftarrow \text{emptylist}$.

2. **Collect client contributions and build the PCD tree:**

- (a) For each i representing a different contribution do the following: (we assume more than $\min_t$ clients contribute)
 - i. Send $\mathcal{K}$ along with a fresh registration ID (rt, g_i, π_i) to client C_i (see Figure 3)
 - ii. Receive the contribution $((\text{ct}_i^{\text{masked}}, \text{ct}_i^{\text{key}}, \text{ct}_i^{\text{seed}}), (\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}}))$ from C_i .
 - iii. Check the contribution's validity: (i) parse ct_i^{key} as $(\text{ct}_0, \text{ct}_1)$ and $\mathcal{X}_i^{\text{PCD}} := (\text{rt}, g_i, u, \text{ct}_1)$ and check ct_1 is consistent; and (ii) Check g_i is a fresh ID and the root rt is correct. Then, append g_i to IDs . (iii) Check Π_i^{PCD} is a valid proof for PCD instance $\mathcal{X}_i^{\text{PCD}}$.
 - iv. If valid, then update the current state as below:

$$\text{encKey} += \text{ct}_i^{\text{key}}, \quad \text{encSum} += \text{ct}_i^{\text{masked}}$$

$$\text{encSeeds.append}(\text{ct}_i^{\text{seed}}), \quad \text{proofs.append}((\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}}))$$

- (b) Run the leaf-decoupled PCD to aggregate contributions into a single instance; the base proofs $(\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}})$ appear at the leaves, and each internal node with children $(\mathcal{X}_b, \Pi_b)_{b \in \{0,1\}}$:
 - i. Verify the PCD proof for each instance.
 - ii. Extract $\{\text{ct}_1^{(b)}\}_{b \in \{0,1\}}$ and $\{g^{(b)}\}_{b \in \{0,1\}}$ from the contributions, and then update

$$g^{\text{sum}} \leftarrow g^{(0)} \cdot g^{(1)}, \quad \text{ct}_1^{\text{sum}} \leftarrow \text{ct}_1^{(0)} + \text{ct}_1^{(1)}.$$

- iii. Output the updated values $(g^{\text{sum}}, \text{ct}_1^{\text{sum}})$.

Let $(\mathcal{X}, \Pi)$ denote the result of accumulating all base PCD proofs via the leaf-decoupled architecture. $\mathcal{X}$ includes the sum of IDs, $\text{com}_{\text{pv}} := g^{\text{sum}}$, and the sum of ciphertexts ct_1^{sum} .

- 3. **Prove uniqueness of IDs:** Let $\text{pv}(\vec{x})$ be the vector committed by com_{pv} . To prove uniqueness of IDs, define $p(\vec{x}) := \text{pv}(\vec{x}) \cdot (1 - \text{pv}(\vec{x}))$ and run a zerocheck to verify that $p(\vec{x})$ vanishes over $\{0, 1\}^N$.
- 4. **Prove minimum participation:** To ensure that at least $\min_t$ clients have contributed, run the sumcheck protocol on $\sum_{x \in \{0,1\}^N} \text{pv}(x) = s$, where $\text{pv}(x)$ has been committed to by com_{pv} , and then check that $s \geq \min_t$.

5. **Decrypt the sum of the contributions:**

- (a) Parse encKey as $(\text{ct}^0, \text{ct}^1)$.
- (b) Run decryption protocol with $\mathcal{D}$ as described in Figure 6:
 - i. Send ciphertext ct^1 , along with proofs (Π, Π_{ID}) .
 - ii. Receive k and sk_{aux} from $\mathcal{D}$.
- (c) Recover masked sum $\text{maskSum} := \sum_{\text{ct}_i^{\text{seed}} \in \text{encSeeds}} \text{PRG.Expand}(\text{E.Dec}(\text{sk}_{\text{aux}}, \text{ct}_i^{\text{seed}}), \ell)$
- (d) Output $\text{KAHE.Dec}(\text{pp}_{\text{KAHE}}, \text{encSum}, k) - \text{maskSum}$.

Figure 5: The server role protocol. Steps 2a and 2b can be run in parallel.

Committee Member Role. The committee members operate in two phases as described below. First, they execute DKG to generate a public key. Later, they assist the server in the decryption process. The formal protocol is provided in Figure 6.

- *Distributed Key Generation.* DKG is run by the committee with the server as the coordinator. Each committee member generates its AHE and auxiliary keys and shares the underlying secret keys with a verifiable secret sharing scheme. These are then aggregated into global public keys. The committee verifies the validity of the aggregated keys and signs them to certify correctness.
- *Decryption.* The role of the decryptor in WillowFold is similar to that in Willow, with one key difference. In Willow, decryptors receive a signature from an external verifier attesting to the correctness of the aggregation, and then decrypt the sum of the keys. In WillowFold, each decryptor instead verifies the aggregation PCD proof on its own; the proof is succinct and efficient to verify. This removes reliance on an external verifier. During decryption, parties use secret shares to reconstruct the keys of any dropped-out members. Finally, once a threshold number of valid partial decryptions is collected, the server can complete the decryption procedure.

6 Security Analysis

We analyze the security of WillowFold. Before that, we justify the need for simulation extractability.

6.1 Ciphertext Correlation Intractability

Each client C_i must prove knowledge of the key k_i , i.e., the plaintext underlying ct_i^{key} , to rule out correlated ciphertext attacks. However, knowledge soundness alone is insufficient: we require the proofs to be non-malleable. For example, if per-client proofs were additively homomorphic, an adversary could sum the ciphertexts ct_i^{key} together with their proofs and thereby mount the attack. To formalize this intuition, we define security against correlated ciphertext attacks via a property we call *ciphertext correlation intractability*. The formal definition appears in Appendix D.1.

At a high level, this property ensures that no adversary, given a set $(c_i, \pi_i)_i$ of AHE ciphertexts—each accompanied by a proof of knowledge of its underlying *random* plaintext m_i —can produce a new ciphertext c^* together with a valid proof of knowledge of its plaintext m^* such that m^* is correlated with the original messages $\{m_i\}_i$. In other words, we require that $(m^*, \{m_i\}_i) \approx_c (m^*, \{b_i\}_i)$ for uniformly random messages $\{b_i\}_i$, where $\approx_c$ denotes *computational indistinguishability*. Note that m_i ’s correspond to KAHE keys in WillowFold. In Appendix D.1, we show that ciphertext correlation intractability holds when AHE is IND-CPA secure and the proofs of plaintext knowledge are simulation extractable (see Appendix C.1 for the formal definition of SE).

6.2 Simulation Security Proofs of WillowFold

We show security of WillowFold in the standard ideal/real simulation paradigm with respect to the secure aggregation functionality in Figure 1. Below, we outline the simulator in both malicious and semi-honest cases. The hybrid argument appears in Appendix D. Let Sim denote the simulator, $\mathcal{A}$ denote the adversary, and H and C be the sets of honest and corrupted clients, respectively.

Malicious Security. For DKG, Sim runs the DKG protocol on behalf of the honest committee members. For aggregation, Sim uses input 0^ℓ and sends contributions on behalf of all honest clients. The adversary $\mathcal{A}$ controls the server and forwards a PCD instance $(\mathcal{X}^{\text{PCD}}, \Pi^{\text{PCD}})$ that includes a ciphertext ct^{sum} and a commitment com_{pv} . Once Sim receives this message, it reprograms the

Each decryptor (committee member) participates in two phases—DKG and decryption—as follows.

Distributed Key Generation

1. Generate keys and secret-share them:

- (a) Each committee member D_j proceeds as follows:
 - i. Sample randomness r_j and r_j^{aux} and generate key pairs:

$$(\text{pk}_j, \text{sk}_j, \pi_j) := \text{AHE.VerifiableKeyGen}(r_j), (\text{pk}_j^{\text{aux}}, \text{sk}_j^{\text{aux}}, \pi_j^{\text{aux}}) := \text{E.VerifiableKeyGen}(r_j^{\text{aux}})$$
 - ii. Secret-shares r_j and r_j^{aux} within the committee with threshold t .
 - iii. Send $(\text{pk}_j, \pi_j, \text{pk}_j^{\text{aux}}, \pi_j^{\text{aux}})$ along with a valid signature σ_j to Coord.
- (b) Coord collects messages up to a timeout T .
- (c) Let $\mathcal{D}$ be the set of members provided valid proofs and signatures. If $|\mathcal{D}| < t$, abort. Otherwise:

$$\text{pk} := \sum_{j \in [m]} \text{pk}_j, \quad \text{pk}^{\text{aux}} := \sum_{j \in [m]} \text{pk}_j^{\text{aux}}.$$

2. Verify the keys and sign them:

- (a) Coord broadcasts $S := \{(\text{pk}_j, \pi_j, \text{pk}_j^{\text{aux}}, \pi_j^{\text{aux}}), \sigma_j\}_{j \in \mathcal{D}}$ within $\mathcal{D}$.
- (b) Each decryptor $D_j \in \mathcal{D}$:
 - i. Check $|S| \geq t$ and proofs π_j and π_j^{aux} and signatures σ_j are valid. Otherwise aborts.
 - ii. Send a signature on $(\text{pk}, \text{pk}^{\text{aux}})$ to Coord.
- (c) Coord collects signatures up to timeout T and defines Σ as the resulting set of signatures.
- (d) If signatures in Σ are valid and $|\Sigma| \geq t$, Coord sends $(\mathcal{D}, \Sigma, \text{pk}, \text{pk}^{\text{aux}})$ to $\mathcal{D}$; otherwise aborts.

Verification and Decryption

1. Verify the proofs, decrypt the ciphertexts, and forward reconstruction shares:

- (a) Receive input $(\text{ct}^1, (\mathcal{X}^{\text{PCD}}, \Pi^{\text{PCD}}), (\mathcal{X}^{\text{ID}}, \Pi^{\text{ID}}))$ from the server. Each decryptor $D_j \in \mathcal{D}$:
 - i. Verify proofs Π^{PCD} and Π^{ID} , and ensure their consistency with ct^1 .
 - ii. Send the partial decryption $\text{pd}_j := \text{AHE.PartialDec}(\text{ct}^1, \text{sk}_j)$.
 - iii. Send shares of r_i^{aux} received in DKG to reconstruct the secret key sk^{aux} .
- (b) Coord collects messages up to timeout T . Let P be the subset of responders. If $|P| < t$, abort.
- (c) Otherwise, Coord reconstructs sk^{aux} . If it doesn't match the pk^{aux} from DKG, abort.

2. Construct secret keys and partial decryptions for inactive decryptors:

- (a) Coord sends P to every decryptor in P .
- (b) Each $D_j \in P$, aborts if $D_j \notin P$. Otherwise sends shares received each $D_k \notin P$ to Coord.
- (c) Coord reconstructs sk_k for each $D_k \notin P$. If it does not match pk_k , abort.
- (d) Coord sends $p_j := \sum_{j \in \mathcal{D}} p_j$ and sk^{aux} to the server.

Figure 6: The decryptor (committee member) role protocol.

random oracle so that the sum of the honest clients' contributions *that the server included in the aggregation* matches the real-world sum. To this end, **Sim** runs the PCD extractor on $(\mathcal{X}^{\text{PCD}}, \Pi^{\text{PCD}})$ to recover the full PCD transcript and the witnesses for leaf nodes and, using this information, **Sim** computes the set S_H of the honest clients that were included by the server in the final aggregate. **Sim** then submits S_H to $\mathcal{F}_{\text{Agg}}$ to obtain the sum of the relevant honest inputs. Finally, **Sim** reprograms the random oracle at the points used by the PRG of one of the honest clients so that the simulated contributions (with input 0^ℓ) match the real-world contributions in sum. Moreover, in the above simulation, **Sim** also ensures that IDs are unique and verifies the PCD proof, therefore playing the role of the verifier.

Semi-Honest Security. Consider a semi-honest server that follows the protocol, but whose memory is observed by the adversary. For setup and DKG, **Sim** runs the protocol on behalf of the server and the committee members. For aggregation, as in the previous case, **Sim** uses input 0^ℓ to simulate the honest clients and additionally runs the server protocol. In this setting, however, **Sim** cannot query $\mathcal{F}_{\text{Agg}}$ for the sum of only the honest clients' inputs; instead, it must provide $\mathcal{F}_{\text{Agg}}$ with the full set of participating clients, including both honest and corrupted ones. To reprogram the random oracle, **Sim** therefore needs to learn the sum of the corrupted clients' inputs and subtract it from the output returned by $\mathcal{F}_{\text{Agg}}$. This can be done by running the extractor for the PCD base proofs received from $\mathcal{A}$; it outputs witnesses that include the inputs chosen by $\mathcal{A}$. **Sim** also checks that the IDs are unique, which can be verified directly from the proofs sent by the adversary.

As discussed in Section 6.1, clients' proofs must satisfy simulation extractability to rule out correlated ciphertext attacks. In Willow, these proofs are instantiated with Bulletproofs [Bün+18], which is simulation extractable [DG23]. In WillowFold, the situation is more subtle, since clients' proofs are aggregated into a single instance via a leaf-decoupled PCD scheme. Below, we elaborate on simulation extractability in the context of PCD schemes.

Simulation Extractability for PCD Schemes. At a high level, a non-interactive argument is simulation extractable if it remains extractable (i.e., knowledge sound) even when the adversary has oracle access to the zero-knowledge simulator and may query the oracle on arbitrarily chosen instances (see Definition C.1). This property implies non-malleability: an adversary cannot forge an accepting proof without knowing a witness, even after observing honestly generated proofs.

Such a definition, however, does not extend directly to PCD schemes. For example, an adversary may query the simulator for a PCD proof and then locally run the next PCD step to obtain a fresh proof. Since this new proof is built on a simulated proof, the PCD transcript may not be fully extractable. A concurrent work [Abd+25] addressed this issue and introduced a notion of simulation extractability for PCD in which the extractor extracts witnesses only from non-simulated PCD nodes, and stops once it reaches a simulated proof. They show that a PCD scheme built from a NARK and an accumulation scheme for the NARK verifier satisfies SE, provided that the NARK is simulation extractable and the accumulation scheme is zero-knowledge and knowledge sound.

WillowFold uses leaf-decoupled PCD in which the argument and accumulation schemes at the internal nodes are not zero-knowledge. In Appendix C.2, we tailor the notion of SE to leaf-decoupled PCDs where the simulator answers only base proof (leaf) queries. We call this notion *leaf-SE*, and show that it holds when base proofs are SE, and the accumulation scheme is knowledge sound.

7 Concrete Instantiation with Spartan+KZH-Fold

WillowFold is generic over any leaf-decoupled PCD scheme that satisfies leaf-SE. In this work, we instantiate WillowFold using a variant of Spartan+KZH-Fold [Kad+25]. KZH-Fold is an accumulation

scheme for the KZH verifier predicate; details of KZH and KZH-Fold appear in Appendix A.3. Spartan+KZH-Fold builds on Spartan+KZH as the underlying SNARK and is compiled into a PCD via the BCLMS compiler [Bün+21a] using KZH-Fold. The main challenge is that Spartan+KZH, as described in [Kad+25], is not zero-knowledge and therefore cannot be simulation extractable.

In this section, we first justify our choice of Spartan+KZH-Fold and outline its advantages over other PCD schemes. We then describe how to obtain a zero-knowledge variant of Spartan+KZH, which we call zk-Spartan+zk-KZH. Finally, we explain how to build a leaf-decoupled PCD from zk-Spartan+zk-KZH using KZH-Fold, and how to show that this scheme satisfies leaf-SE.

7.1 Properties of Spartan+KZH-Fold

We discuss the properties of our choice next.

Sublinearity. Most state-of-the-art PCD schemes built from linear-size accumulation—such as Nova [KST22], HyperNova [KS24], and Protostar [BC23]—have proof size and verification time linear in the circuit size N . In contrast, Spartan+KZH-Fold achieves sublinear proof size and verification time, namely $O(k \cdot N^{1/k})$ for an arbitrary parameter k , which becomes $O(\log N)$ when $k = \log N$. To illustrate the concrete impact, consider the client circuit in WillowFold, which has roughly 1.4M constraints. For this circuit, Nova yields a proof of about 120 MB and verification time of about 7 seconds. In our benchmarks, Spartan+KZH-Fold produces proofs below 50 KB with verification time around 470 ms, while the prover is only about $3\times$ slower than Nova. Linear-size PCDs therefore impose substantial client-server communication overhead, since clients must transmit large proofs, and both the server and the committee incur high costs when verifying them.

Low-Cost Zero-Knowledge. Spartan+KZH (that is, the SNARK underlying Spartan+KZH-Fold) can be made zero-knowledge with only sublinear overhead (see Section 7.2). This efficiency stems from the sublinear proof size of Spartan+KZH, which in turn means that masking the witness requires only a sublinear amount of randomness. For example, HyperNova [KS24] follows a similar approach by sampling a random blinding accumulator and masking the final proof with it; however, since its accumulator is linear-sized, their scheme requires a linear amount of randomness.

In terms of concrete efficiency, using *linear-size* randomness can increase prover runtime by roughly an order of magnitude. This effect is particularly pronounced in WillowFold, where most witness entries are binary. MSM-based schemes such as Nova, HyperNova, and KZH can exploit this low-weight structure, making commitments to low-weight witnesses about $10\times$ faster than commitments to arbitrary witnesses. However, masking the witness with linear-size random field elements destroys this low-weight property, leading to an additional $\approx 10\times$ overhead.

A different zero-knowledge accumulation scheme that uses only sublinear randomness is based on KZG polynomial commitments [Bün+21a]. However, unlike KZH, KZG does not exploit low-weight witnesses and, as noted above, similarly incurs an $\approx 10\times$ increase in prover (client) cost.

Simulation Extractability. We further show that the zero-knowledge variant of Spartan+KZH satisfies simulation extractability (see Section 7.3). Prior works such as Nova are malleable, and while mitigations exist, they apply only to the final proof [NBS] rather than to PCD leaves, which is what we require in our setting. We believe the same limitation applies to HyperNova.

Support of Commit-and-Prove. As we discuss in Section 8.1, achieving concrete efficiency requires the PCD scheme at the leaves to support the *commit-and-prove* paradigm. Concretely, the witness of the PCD circuit is a multilinear polynomial $w(\vec{Y}, \vec{X})$ that can be decomposed into two parts $w_0(\vec{X}) := w(0, \vec{X})$ and $w_1(\vec{X}) := w(1, \vec{X})$, such that:

$$w(y, \vec{X}) = (1 - y) \cdot w_0(\vec{X}) + y \cdot w_1(\vec{X}).$$

Here, $w_0(\vec{X})$ is computed independently, whereas $w_1(\vec{X})$ is computed from $w_0(\vec{X})$ and a verifier challenge $\vec{y}$ that fixes $\vec{Y} := \vec{y}$. An advantage of Spartan+KZH is that, like Spartan, it naturally supports the commit-and-prove paradigm. In standard Spartan, the prover provides the verifier with an oracle (i.e., a commitment) to the full witness $w(\vec{X})$. In the commit-and-prove variant, the prover first sends an oracle to $w_0(\vec{X})$. The verifier then responds with a challenge, after which the prover computes $w_1(\vec{X})$ and sends an oracle to $w_1(\vec{X})$. Given oracle access to both $w_0(\vec{X})$ and $w_1(\vec{X})$, the verifier can efficiently check their consistency with the oracle to $w(\vec{X})$.

7.2 Zero-Knowledge Spartan+KZH

Spartan+KZH builds on Spartan polynomial IOP and KZH polynomial commitment. As shown in [Set20; Che+23], Spartan can be made zero-knowledge via a zero-knowledge sumcheck [Xie+19]. IronDict [Haf+25], also presents a zero-knowledge variant of the KZH commitment scheme. Putting these together, composing zero-knowledge Spartan (zk-Spartan) with zero-knowledge KZH (zk-KZH) would yield a zero-knowledge SNARK via a standard compilation [Chi+20]. This transformation requires a logarithmic amount of randomness, since both zk-Spartan and zk-KZH only use logarithmic randomness.

The main caveat is that the zero-knowledge sumcheck protocol from prior work [Xie+19] requires committing to higher-degree multivariate polynomials, rather than multilinear ones. However, KZH is designed for multilinear polynomial commitments. Below, we elaborate on this issue.

Zero-Knowledge Sumcheck. Consider a sumcheck instance $H := \sum_{\vec{x} \in \{0,1\}^\ell} f(\vec{x})$, where $f(\vec{X})$ is an ℓ -variate polynomial of individual degree d . To achieve zero knowledge, the prover masks $f(\vec{X})$ with a random polynomial $r(\vec{X})$ of the same degree and sends $G := \sum_{\vec{x} \in \{0,1\}^\ell} r(\vec{x})$ and a commitment to $r(\vec{X})$. Upon receiving a challenge ρ , the parties run a sumcheck protocol on

$$G + \rho \cdot H = \sum_{\vec{x} \in \{0,1\}^\ell} (r(\vec{x}) + \rho \cdot f(\vec{x})).$$

As shown by Xie et al. [Xie+19], it suffices for the masking polynomial $r(\vec{X})$ to be sampled from a highly structured family, where $r(\vec{X}) := r_1(X_1) + r_2(X_2) + \dots + r_\ell(X_\ell)$, such that each $r_i(X_i)$ is a univariate polynomial of degree d . This representation uses only $d\ell + 1$ random coefficients.

The barrier is that, even though the required randomness is only logarithmic, the polynomial commitment scheme must support degree- d polynomials in order to commit to $r(\vec{X})$. In Spartan, some sumcheck instances have $d = 2$ or $d = 3$, whereas KZH supports only multilinear polynomials.

To resolve this issue, one may use a generic transformation from multivariate to multilinear polynomials [Zha+24], however, this would blow up the polynomial size to $O((d+1)^\ell)$. Here, we exploit the structure of the randomness, and propose a method to commit to $r(\vec{X})$ with only $O(d\ell)$ overhead. The scheme is given in Figure 7.

Theorem 1. Π_{commit} in Figure 7 satisfies zero-knowledge and knowledge soundness.

Proof. The scheme is zero-knowledge because PCS is zero-knowledge, and each coefficient of $r(\vec{X})$ has been masked with a random coefficient of $h(\vec{X})$. Soundness follows from the soundness of PCS, the soundness of sumcheck, and from the fact that, over the choice of ρ , for any $y^* \neq r(\vec{x})$, it holds that $\rho \cdot y^* + y_h \neq \rho \cdot r(\vec{x}) + h(\vec{x})$ except for a negligible probability $1/|\mathbb{F}|$. Finally, the commitment is extractable since PCS commitments are extractable, and $r(\vec{x})$ can be extracted from c . $\square$

Let PCS be a zero-knowledge multilinear polynomial commitment scheme with public parameters pp_{PCS} . Moreover, let $r(\vec{X}) = v_0 + \sum_{i \in [\ell], j \in [d]} v_{i,j} X_i^j$ denote the sumcheck masking polynomial.

Commitment:

1. Set $\tilde{v}$ to be the multilinear extension of $(v_0, v_{1,1}, \dots, v_{\ell,d})$.
2. Return $\text{PCS.Com}(\text{pp}_{\text{PCS}}, \tilde{v}, r) \rightarrow c$.

Opening at point $\vec{x}$:

1. Sample random $h(\vec{X}) = u_0 + \sum_{i \in [\ell], j \in [d]} u_{i,j} X_i^j$.
2. Set $\tilde{u}$ to be the multilinear extension of $(u_0, u_{1,1}, \dots, u_{\ell,d})$.
3. Run $\text{PCS.Com}(\text{pp}_{\text{PCS}}, \tilde{u}, r) \rightarrow c_h$.
4. Send c_h , $y_h = h(\vec{x})$, and $y = r(\vec{x})$, and receive a challenge ρ .
5. Set $\tilde{x}^*$ to be the multilinear extension of $(x_i^j)_{i,j}$.
6. Run the sumcheck protocol on

$$\rho y + y_h = \sum_{\vec{b} \in \{0,1\}^{\log \lceil d\ell+1 \rceil}} (\rho \tilde{v}(\vec{b}) + \tilde{u}(\vec{b})) \cdot \tilde{x}^*(\vec{b}),$$

which ends with evaluations $p := \tilde{v}(\vec{s})$, $q := \tilde{u}(\vec{s})$, and $t := \tilde{x}^*(\vec{s})$ for a random challenge point $\vec{s}$.

7. Run $\text{PCS.Open}(\text{pp}_{\text{PCS}}, \tilde{v}(\vec{X}), \vec{s}) \rightarrow \pi_r$ and $\text{PCS.Open}(\text{pp}_{\text{PCS}}, \tilde{u}(\vec{X}), \vec{s}) \rightarrow \pi_h$.
8. Return the sumcheck messages, π_r and π_h .

Verification:

1. Verify the sumcheck messages and their consistency with y and y_h .
2. Run $\text{PCS.Verify}(\text{pp}_{\text{PCS}}, c, \vec{s}, p)$ and $\text{PCS.Verify}(\text{pp}_{\text{PCS}}, c_h, \vec{s}, q)$, and check if they accept.
3. Check if $t = \tilde{x}^*(\vec{s})$.

Figure 7: Commitment to sumcheck masking randomness via a multilinear PCS.

7.3 Simulation Extractibility of Spartan+KZH-Fold

Recall that zk-Spartan+zk-KZH is a non-interactive zero-knowledge argument with updatable SRS. Prior work [Gan+22] showed that such an argument is simulation extractable if it has three properties: (i) special soundness, (ii) trapdoor-less zero-knowledge, and (iii) unique response (see Appendix C.1). To prove simulation extractability of zk-Spartan+zk-KZH, each of these must be shown.

Special soundness of zk-Spartan+zk-KZH follows from the soundness of zk-Spartan [Set20; Che+23] and the special soundness of zk-KZH [Kad+25; Haf+25]. The latter was previously shown only in the standalone setting; in Lemma C.2, we extend the prior proofs to the updatable-SRS setting.

Trapdoor-Less Zero-Knowledge. This property requires the zero-knowledge simulator of the scheme to reprogram the random oracle in at most one round and not use the setup trapdoor. The

simulator of zk-Spartan+zk-KZH invokes the simulator of zero-knowledge sumcheck and zk-KZH, each for one time. The simulator for sumcheck is information-theoretic, and neither reprograms the random oracle nor uses a trapdoor. The simulator of KZH, reprograms the random oracle only when a commitment is opened. Since all openings in zk-Spartan+zk-KZH occur in the final round, zk-Spartan+zk-KZH is trapdoor-less zero-knowledge (see Lemma C.5 for formal proof).

Unique Response. This property ensures that there is a particular round r between the prover and the verifier such that if two proofs π and π^* have identical messages up to round r , then $\pi = \pi^*$. We further require that this round coincide with the round in which the simulator reprograms the random oracle—i.e., the final round of Spartan+KZH. As discussed above, it suffices to show that for a given KZH commitment and evaluation, the opening is unique. We prove this in Lemma C.6.

As mentioned in Section 6.2, a leaf-decoupled PCD built from a simulation extractable SNARK and a knowledge sound accumulation scheme satisfies leaf-SE. Prior work [Kad+25] showed knowledge soundness of the KZH-Fold accumulation scheme. Thus, SE of zk-Spartan+zk-KZH implies that leaf-decoupled Spartan+KZH-Fold satisfies leaf-SE. Formal proofs appear in Appendix C.2.

Remark 2. *The KZH-Fold accumulation scheme for Spartan+KZH commitments can be adapted to zk-Spartan+zk-KZH with only minor modifications: the Spartan PIOP messages are handled exactly as before. For zk-KZH, the accumulator performs only a constant number of additional group operations beyond those in KZH-Fold. We defer full details to Appendix B.*

8 Implementation and Evaluation

Our implementation of WillowFold builds on the Arkworks [con22], KZH-Fold⁵ and Willow⁶ codebases. We instantiate KZH-Fold with KZH-3 and use a half-pairing cycle of curves BN254 and Grumpkin. Our codebase will be publicly available soon. We use a high-performance Google Cloud machine (c4-highcpu-288) as the server, with 576 GB of RAM and 288 vCPUs. For the client and committee members, we use a MacBook Air M3 with an 8-core CPU and 8 GB of RAM. In our experiments, the server requires at most 20 GB of RAM to perform aggregation. We particularly selected a high-performance machine primarily for its CPU power, otherwise the aggregation process itself is not memory-intensive. Since we employ the same lattice-based schemes, AHE and KAHE, as Willow, our choice of lattice parameters are identical to Willow. For further details, we refer the reader to Section 7 and Table 2 of Willow [BC+25].

8.1 Optimizations

Smallness Test. For smallness tests of the form $x \leq L$ with a bound $L \ll |\mathbb{F}|$, we proceed as follows. First, we embed the bit-decomposition of L in the circuit to be used across all tests. For each x , the prover gives its bit-decomposition $b_0, b_1, \dots, b_{\lceil \log L \rceil - 1}$. The circuit checks $b_j \cdot (1 - b_j) = 0$ for all j and that $x = \sum_j b_j 2^j$, where the powers of two are hardcoded in the circuit. By a lexicographic comparison of the bit-decompositions of x and L , we can verify the test. This technique requires a $\lceil \log L \rceil$ -sized witness, while the generic arkworks smallness test has a $\log |\mathbb{F}|$ -sized witness.

Efficient Non-Native Arithmetic. We use a power-of-two cyclotomic ring $R = \mathbb{W}[X]/(X^N + 1)$ in our AHE scheme. Elements of R are polynomials with coefficients in a finite field $\mathbb{W}$, which differs from the field $\mathbb{F}$ used by our KZH-Fold circuit (e.g., the scalar field of BN254). The field $\mathbb{W}$ is much

⁵https://github.com/h-hafezi/kzh_fold

⁶<https://github.com/google/secure-aggregation>

n	One-time Setup		Server					
	KZH-Fold	Merkle	Verify time (per client)	Contribution Proof Size	Aggregate (per client)	Proof of uniqueness	IDs	Setup
2^{10}	2.55 s	32 ms	470 ms	38 KB	3.8 s	0.11 s	40 KB	16.3 MB
2^{17}	2.8 s	4.03 s	475 ms	38 KB	3.8 s	0.33 s	4.22 MB	16.3 MB
2^{23}	10.8 s	255 s	478 ms	38 KB	3.8 s	3.2 s	310 MB	16.3 MB

Table 2: Microbenchmarks for WillowFold Setup and Server for number of clients $n \in \{2^{10}, 2^{17}, 2^{23}\}$. We break up the setup cost into the KZH-fold PCD setup and the ID Merkle Tree generation. Reported sizes assume compression of group elements.

smaller, typically of size 2^{64} – 2^{96} (see Table 2 in Willow). Since elements of $\mathbb{W}$ are represented in $\mathbb{F}$, the circuit includes range checks that values are less than $|\mathbb{W}|$. To enforce modular correctness, each modular operation $a \star b = c$ is expressed as $a \star b - c = |\mathbb{W}| \cdot q$, with $a, b, c, q < |\mathbb{W}|$.

The main operation in the circuit is a ring multiplication, expressed as a polynomial product $u(r) \cdot v(r) = w(r)$, where u, v, w have coefficients in $\mathbb{W}$ and degree n . Thus, we design an efficient interpolation method for *non-native* polynomials. As the first step, we compute powers $\{r^i\}_{i \in [n]}$ recursively using $r^i = r^{i-1} \cdot r + |\mathbb{W}| \cdot m_i$, where m_i is the quotient from division. Since r^{i-1} is already known to be small from the previous step, we only need to check $r^i, m_i < |\mathbb{W}|$. We then evaluate $u(r) = \sum u_i r^i$ as an inner product between $(1, r, \dots, r^n)$ and $(u_0, u_1, \dots, u_n)$. We make two observations that boost our efficiency. Let 2^ℓ be the smallest power of two greater than $|\mathbb{W}|$; it suffices to prove $r^i, u_i < 2^\ell$. Since $n \cdot |\mathbb{W}|^2 < n \cdot 4^\ell < |\mathbb{F}|$, we can safely compute the inner product (or polynomial evaluation) over $\mathbb{F}$ since no overflow would happen, and then perform a single modular reduction by $\mathbb{W}$ at the end. Importantly, checking a value is less than 2^ℓ is much more efficient in the circuit than checking an arbitrary value, since it doesn’t require a lexicographic comparison.

Probabilistic Verification of Ring Operations. The client circuit requires computing the product of two ring elements $\mathbf{u}, \mathbf{v} \in \mathbb{W}/(X^N + 1)$, which is performed using the *negacyclic matrix*. This is the method currently used in Willow to compute $\mathbf{v} \cdot \mathbf{u}$ within an arithmetic circuit. However, this approach is highly inefficient: with $N = 4096$, it yields a circuit with roughly 16 million constraints. We propose an alternative method that leverages the underlying polynomial structure. Let $\mathbf{w} = \mathbf{v} \cdot \mathbf{u}$ be the product of the corresponding ring elements. By evaluating the underlying polynomials of $\mathbf{v}, \mathbf{u}$ and $\mathbf{w}$ at a randomly chosen point $r \in \mathbb{W}$, we can check the equality probabilistically: if $\mathbf{w}(r) = \mathbf{v}(r) \cdot \mathbf{u}(r)$, then with overwhelming probability we have that $\mathbf{w} = \mathbf{v} \cdot \mathbf{u}$.⁷

This reduces the verification cost from quadratic to linear in the number of constraints. To securely obtain the randomness r , we need to hash the polynomials (ring elements) $\mathbf{v}, \mathbf{u}, \mathbf{w}$ and derive r from the output. However, hashing these polynomials directly is prohibitively expensive: even with a zk-friendly hash function such as Poseidon [Gra+21], which requires about 100 constraints per element, hashing three ring elements of length 4096 yields nearly 1.2 million constraints. To avoid this cost, we adopt the *commit-and-prove* paradigm, i.e., the circuit is provided with commitments to $\mathbf{v}, \mathbf{u}$ and $\mathbf{w}$ and derives the challenge r by hashing the commitments.

Remark 3 (Use of Lookup Table Arguments). One common approach to reducing the cost of non-field-friendly operations—such as binary operations—is the use of lookup tables [GW20; STW24]. Lookup tables replace these non-native operations with lookups in precomputed tables,

⁷Since $|\mathbb{W}| \leq 2^{96}$, we repeat the check at two independent challenges $r_1, r_2 \in \mathbb{W}$ to achieve 128-bit soundness.

(n, ℓ)	Client			Committee Member			
	Setup	Enc	Prove	Setup	PCD Verify / Size	Zerocheck Verify / Size	Decryption
$(2^{10}, 10^3)$	64.5 MB	2.3 ms	2.9 s	35 KB	910 ms	47 ms	74 ms
$(2^{10}, 10^5)$		8.2 ms			48 KB	1.05 KB	148 ms
$(2^{10}, 10^7)$		296 ms					297 ms
$(2^{17}, 10^3)$	64.5 MB	2.3 ms	2.9 s	39 KB	910 ms	81 ms	84 ms
$(2^{17}, 10^5)$		8.2 ms			48 KB	3.0 KB	148 ms
$(2^{17}, 10^7)$		296 ms					283 ms
$(2^{23}, 10^3)$	64.5 MB	2.3 ms	3.0 s	48 KB	910 ms	104 ms	84 ms
$(2^{23}, 10^5)$		8.2 ms			48 KB	7.6 KB	144 ms
$(2^{23}, 10^7)$		296 ms					324 ms

Table 3: Microbenchmarks of WillowFold’s Client and Committee for input lengths $\ell \in \{10^3, 10^5, 10^7\}$ and number of clients $n \in \{2^{10}, 2^{17}, 2^{23}\}$. The values filling out multiple cells are independent of ℓ . For the client we measure the setup/proving key size, the encryption time, and the proving time. For the committee we report the setup size, the time to verify the PCD proof, the ZeroCheck proof and their sizes.

thereby reducing the circuit size and improving efficiency. However, applying lookup tables efficiently in recursive settings, particularly for PCDs, remains an open challenge. Existing approaches, such as FLI [GM24] and Protostar [BC23] do not offer efficient PCD constructions—the prover’s cost to aggregate PCD proofs is proportional to the size of the table. Another work, nLookup [KS24], assumes that the witness is already given in bit-decomposed form, thereby increasing the circuit size. While nLookup is well-suited to complex operations, our setting primarily involves simple operations expressed via bit decomposition. Hence, nLookup does not improve efficiency in our setting. For a review on lookup-table arguments in recursive settings, we refer the reader to [Haf+26, Appendix H].

8.2 Setup and Registration Costs

Our setup (excluding DKG) comprises two main phases: (i) PCD setup (i.e., the variant of Spartan+KZH-Fold), and (ii) generating client IDs and the Merkle tree over them. The KZH and KZH-Fold setups are *trusted*, analogous to KZG [KZG10], and thus must be carried out via a multi-party ceremony [WCB25]. Nevertheless, this requirement does not hinder practicality. As shown in Table 2, a single server can complete the KZH-related setup in under 11 seconds for $n = 2^{23}$ clients. Hence, a sequential SRS ceremony with, e.g., 10 participants is feasible, with each party having 11 seconds of work.

KZH-Fold Setup. Due to leaf decoupling, we require two separate KZH-Fold setups: one for the client circuit and one for the server circuit. An advantage of KZH-Fold is that a larger setup can be used to derive smaller ones—thus, we can run the client setup and then derive the server setup.

Moreover, each role needs to store only a subset of the public parameters. In particular, each client stores the proving key for the KZH-Fold setup corresponding to the client circuit. The server stores the proving key for the server circuit and the verification key for the client circuit. Committee

members store the verification keys for both the client and server circuits.

Construct Merkle Tree of IDs. The second setup component commits to unit vectors via KZH to generate client IDs, and then the server constructs a Merkle tree over these IDs. Since clients must later prove Merkle inclusion within their circuits, the Merkle hash must be *zk-friendly*, i.e., efficient to evaluate inside an arithmetic circuit. In our implementation, we use MiMC [Alb+16] with exponent 3 and 91 rounds, which provides approximately 128-bit security over the BN254 field.

8.3 Client, Server, and Decryptor Costs

Client Costs. The client’s main cost is to construct a zero-knowledge proof of knowledge of the secret key used in the AHE scheme. Our client circuit consists of 2^{21} R1CS constraints, padded to a power of two to satisfy the Spartan PIOP requirements. Notably, the witness is very sparse: over 90% of a valid witness consists of binary field elements. As shown in Table 3, prover time increases only slightly as the number of clients grows. This is because Merkle inclusion—the only component that depends on the number of clients—accounts for only a small fraction of the overall circuit; for example, it contributes roughly 30k constraints even for one billion clients. We note a subtle limitation of our protocol: each client must store an SRS of size 64.5 MB.

Server Costs. The server performs three main tasks: (i) it aggregates the clients’ contributions, i.e., aggregates the ciphertexts and builds the PCD tree; (ii) once aggregation is complete, it proves that each client’s contribution is used at most once, and finally, (iii) it performs the decryption.

- *Per-Client Cost.* Each client submits a PCD proof along with several ciphertexts to the server. Since ciphertext aggregation is computationally negligible, the main server workload per client involves (i) verifying the client’s PCD proof and (ii) aggregating it into the ongoing PCD proof. The server circuit, i.e., internal nodes of the PCD tree, consists of 2^{16} R1CS constraints.
- *Proof of Uniqueness.* After aggregating all client contributions, the final PCD proof includes a commitment to the participation vector. The server runs a zerocheck to prove the underlying vector is binary. Concretely, this step is very efficient, taking ≈ 3 seconds for 2^{23} clients.

Committee Cost. Each committee member verifies the protocol execution to ensure correctness before acting as a decryptor. This verification cost is sublinear in the number of clients and independent of the input dimension; as shown in Table 3, it grows only logarithmically. Committee members are also responsible for decryption, which has the same cost as in Willow.

8.4 Concrete Comparison with Prior Work

Our protocol is the first single-server aggregation scheme with dynamic clients and sublinear committee work, achieved by shifting verification from a trusted verifier to the untrusted server. This raises per-client server cost but is worthwhile, since committee members are typically lightweight end devices while the server can be powerful.

Willow. The closest to our work for comparison is Willow, since we build our scheme on Willow and only modify its underlying proof system. In Table 4, we give a concrete comparison of Willow with WillowFold for $N = 2^{23}$ clients. Since both schemes employ similar encryption schemes and have similar decryptor overhead, we exclude the decryptor role from this comparison.

While Willow achieves lower client and server costs, its high verification overhead for committee members forces an unfavorable trade-off: either Willow needs to use a large committee—weakening

Scheme	Client Proof Gen.	Server per Client	Total Verifier	Proof Size
Willow	0.8 s	0.2 s	460 h	15 GB
WillowFold	2.9 s	4.3 s	1.0 s	57 KB

Table 4: Comparison of Willow and WillowFold for aggregating $N = 2^{23}$ clients. The client’s proof generation time, the server’s per-client aggregation time, the total verifier time, and the total proof size are shown.

trust assumptions by requiring an honest majority among more parties—or one keeps the committee small and incurs high per-member computation overhead, which limits scalability.

WillowFold at Scale. We conclude by noting that the costs incurred by the client, the committee, and the server (per-client) all grow only logarithmically with the number of users. In particular, each client must prove $\log N$ hash computations within its circuit to verify its identity. Even for $N = 10^9$, this subcircuit requires less than 30k constraints—insignificant compared to the roughly 1.4 million constraints already present in the client’s circuit. This shows that the additional overhead introduced by scaling to large user populations is minimal.

Acknowledgment

Hossein Hafezi carried out this work while being partially hosted by Martin Kleppmann at the University of Cambridge and Bart Preneel at KU Leuven. We thank anonymous reviewers of IEEE S&P 2026 for their constructive comments. We thank Matteo Campanelli and Hamidreza Khoshakhlagh for the discussion on our notion of SE. This work was supported in part by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02.

References

- [Abd+25] B. Abdolmaleki, M. Campanelli, Q. Dao, and H. Khoshakhlagh. *On the Simulation-Extractability of Proof-Carrying Data*. Cryptology ePrint Archive, Report 2025/2037. 2025. URL: <https://eprint.iacr.org/2025/2037>.
- [Add+22] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou. “Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares”. In: *SCN 22: 13th International Conference on Security in Communication Networks*. Ed. by C. Galdi and S. Jarecki. Vol. 13409. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Cham, Switzerland, 2022, pp. 516–539. DOI: [10.1007/978-3-031-14791-3_23](https://doi.org/10.1007/978-3-031-14791-3_23).
- [Alb+16] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. “MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity”. In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer Berlin Heidelberg, Germany, 2016, pp. 191–219. DOI: [10.1007/978-3-662-53887-6_7](https://doi.org/10.1007/978-3-662-53887-6_7).
- [APS15] M. Albrecht, R. Player, and S. Scott. “On the concrete hardness of Learning with Errors”. In: *Journal of Mathematical Cryptology* 9 (Oct. 2015). DOI: [10.1515/jmc-2015-0016](https://doi.org/10.1515/jmc-2015-0016).
- [BC23] B. Bünz and B. Chen. “Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols”. In: *Advances in Cryptology – ASIACRYPT 2023, Part II*. Ed. by J. Guo and R. Steinfeld. Vol. 14439. Lecture Notes in Computer Science. Guangzhou, China: Springer, Singapore, Singapore, 2023, pp. 77–110. DOI: [10.1007/978-981-99-8724-5_3](https://doi.org/10.1007/978-981-99-8724-5_3).
- [BC+25] J. Bell-Clark, A. Gascón, B. Li, M. Raykova, and P. Schoppmann. “Willow: Secure Aggregation with One-Shot Clients”. In: *Advances in Cryptology – CRYPTO 2025, Part VIII*. Ed. by Y. T. Kalai and S. F. Kamara. Vol. 16007. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2025, pp. 285–318. DOI: [10.1007/978-3-032-01913-4_9](https://doi.org/10.1007/978-3-032-01913-4_9).
- [Bel+20] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. “Secure Single-Server Aggregation with (Poly)Logarithmic Overhead”. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by J. Ligatti, X. Ou, J. Katz, and G. Vigna. Virtual Event, USA: ACM Press, 2020, pp. 1253–1269. DOI: [10.1145/3372297.3417885](https://doi.org/10.1145/3372297.3417885).
- [Bel+23] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun. “ACORN: Input Validation for Secure Aggregation”. In: *USENIX Security 2023: 32nd USENIX Security Symposium*. Ed. by J. A. Calandrino and C. Troncoso. Anaheim, CA, USA: USENIX Association, 2023, pp. 4805–4822. URL: <https://www.usenix.org/conference/usenixsecurity23/presentation/bell>.
- [BGH19] S. Bowe, J. Grigg, and D. Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019. URL: <https://eprint.iacr.org/2019/1021>.

- [Bit+13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. “Recursive composition and bootstrapping for SNARKS and proof-carrying data”. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. STOC ’13. Association for Computing Machinery, 2013, 111–120. ISBN: 9781450320290. DOI: [10.1145/2488608.2488623](https://doi.org/10.1145/2488608.2488623). URL: <https://doi.org/10.1145/2488608.2488623>.
- [Bon+17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. “Practical Secure Aggregation for Privacy-Preserving Machine Learning”. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. Dallas, TX, USA: ACM Press, 2017, pp. 1175–1191. DOI: [10.1145/3133956.3133982](https://doi.org/10.1145/3133956.3133982).
- [Bün+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2018, pp. 315–334. DOI: [10.1109/SP.2018.00020](https://doi.org/10.1109/SP.2018.00020).
- [Bün+21a] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. “Proof-Carrying Data Without Succinct Arguments”. In: *Advances in Cryptology – CRYPTO 2021, Part I*. Ed. by T. Malkin and C. Peikert. Vol. 12825. Lecture Notes in Computer Science. Virtual Event: Springer, Cham, Switzerland, 2021, pp. 681–710. DOI: [10.1007/978-3-030-84242-0_24](https://doi.org/10.1007/978-3-030-84242-0_24).
- [Bün+21b] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. “Proofs for Inner Pairing Products and Applications”. In: *Advances in Cryptology – ASIACRYPT 2021, Part III*. Ed. by M. Tibouchi and H. Wang. Vol. 13092. Lecture Notes in Computer Science. Singapore: Springer, Cham, Switzerland, 2021, pp. 65–97. DOI: [10.1007/978-3-030-92078-4_3](https://doi.org/10.1007/978-3-030-92078-4_3).
- [CFQ19] M. Campanelli, D. Fiore, and A. Querol. “LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. Association for Computing Machinery, 2019, 2075–2092. ISBN: 9781450367479. DOI: [10.1145/3319535.3339820](https://doi.org/10.1145/3319535.3339820). URL: <https://doi.org/10.1145/3319535.3339820>.
- [CFS17] A. Chiesa, M. A. Forbes, and N. Spooner. *A Zero Knowledge Sumcheck and its Applications*. Cryptology ePrint Archive, Report 2017/305. 2017. URL: <https://eprint.iacr.org/2017/305>.
- [CGB] H. Corrigan-Gibbs and D. Boneh. “Prio: private, robust, and scalable computation of aggregate statistics”. In: 259–282. ISBN: 9781931971379.
- [Che+23] B. Chen, B. Bünz, D. Boneh, and Z. Zhang. “HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates”. In: *Advances in Cryptology – EUROCRYPT 2023, Part II*. Ed. by C. Hazay and M. Stam. Vol. 14005. Lecture Notes in Computer Science. Lyon, France: Springer, Cham, Switzerland, 2023, pp. 499–530. DOI: [10.1007/978-3-031-30617-4_17](https://doi.org/10.1007/978-3-031-30617-4_17).
- [Chi+20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward. “Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS”. In: *Advances in Cryptology – EUROCRYPT 2020, Part I*. Ed. by A. Canteaut and Y. Ishai. Vol. 12105. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Cham, Switzerland, 2020, pp. 738–768. DOI: [10.1007/978-3-030-45721-1_26](https://doi.org/10.1007/978-3-030-45721-1_26).

- [Cho+22] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten. “EIFFeL: Ensuring Integrity for Federated Learning”. In: *ACM CCS 2022: 29th Conference on Computer and Communications Security*. Ed. by H. Yin, A. Stavrou, C. Cremers, and E. Shi. Los Angeles, CA, USA: ACM Press, 2022, pp. 2535–2549. DOI: [10.1145/3548606.3560611](https://doi.org/10.1145/3548606.3560611).
- [con22] arkworks contributors. *arkworks zkSNARK ecosystem*. 2022. URL: <https://arkworks.rs>.
- [CT10] A. Chiesa and E. Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature Cards”. In: *ICS 2010: 1st Innovations in Computer Science*. Ed. by A. C.-C. Yao. Tsinghua University, Beijing, China: Tsinghua University Press, 2010, pp. 310–331.
- [DG23] Q. Dao and P. Grubbs. “Spartan and Bulletproofs are Simulation-Extractable (for Free!)” In: *Advances in Cryptology – EUROCRYPT 2023, Part II*. Ed. by C. Hazay and M. Stam. Vol. 14005. Lecture Notes in Computer Science. Lyon, France: Springer, Cham, Switzerland, 2023, pp. 531–562. DOI: [10.1007/978-3-031-30617-4_18](https://doi.org/10.1007/978-3-031-30617-4_18).
- [DRZ20] V. Daza, C. Ràfols, and A. Zacharakis. “Updateable Inner Product Argument with Logarithmic Verifier and Applications”. In: *Public-Key Cryptography – PKC 2020*. Ed. by A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas. Cham: Springer International Publishing, 2020, pp. 527–557. ISBN: 978-3-030-45374-9.
- [FL] P. T. S. with Federated Learning. <https://research.google/blog/predicting-text-selections-with-federated-learning/>.
- [Gan+22] C. Ganesh, H. Khoshakhlagh, M. Kohlweiss, A. Nitulescu, and M. Zajac. “What Makes Fiat-Shamir zkSNARKs (Updatable SRS) Simulation Extractable?” In: *SCN 22: 13th International Conference on Security in Communication Networks*. Ed. by C. Galdi and S. Jarecki. Vol. 13409. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Cham, Switzerland, 2022, pp. 735–760. DOI: [10.1007/978-3-031-14791-3_32](https://doi.org/10.1007/978-3-031-14791-3_32).
- [GM24] A. Garreta and I. Manzur. “FLI: Folding Lookup Instances”. In: *Advances in Cryptology – ASIACRYPT 2024, Part V*. Ed. by K.-M. Chung and Y. Sasaki. Vol. 15488. Lecture Notes in Computer Science. Kolkata, India: Springer, Singapore, Singapore, 2024, pp. 402–435. DOI: [10.1007/978-981-96-0935-2_13](https://doi.org/10.1007/978-981-96-0935-2_13).
- [Gra+21] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schafneggger. “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems”. In: *USENIX Security 2021: 30th USENIX Security Symposium*. Ed. by M. Bailey and R. Greenstadt. USENIX Association, 2021, pp. 519–535. URL: <https://www.usenix.org/conference/usenixsecurity21/presentation/grassi>.
- [Gro16] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: *Advances in Cryptology – EUROCRYPT 2016, Part II*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9666. Lecture Notes in Computer Science. Vienna, Austria: Springer Berlin Heidelberg, Germany, 2016, pp. 305–326. DOI: [10.1007/978-3-662-49896-5_11](https://doi.org/10.1007/978-3-662-49896-5_11).
- [GW20] A. Gabizon and Z. J. Williamson. *plookup: A simplified polynomial protocol for lookup tables*. Cryptology ePrint Archive, Report 2020/315. 2020. URL: <https://eprint.iacr.org/2020/315>.
- [Haf+25] H. Hafezi, A. Shirzad, B. Bünz, and J. Bonneau. *IronDict: Transparent Dictionaries from Polynomial Commitments*. Cryptology ePrint Archive, Report 2025/1580. 2025. URL: <https://eprint.iacr.org/2025/1580>.

- [Haf+26] H. Hafezi, G. Anthoine, M. Campanelli, and D. Fiore. “SoK: Lookup Table Arguments”. In: *FC ’26: Proceedings of the 30th International Conference on Financial Cryptography and Data Security*. St. Kitts, 2026. URL: <https://eprint.iacr.org/2025/1876>.
- [Kad+25] G. Kadianakis, A. Zapico, H. Hafezi, and B. Bünz. “KZH-Fold: Accountable Voting from Sublinear Accumulation”. In: *ACM CCS 2025: 32nd Conference on Computer and Communications Security*. Ed. by C.-Y. Huang, J.-C. Chen, S.-P. Shieh, D. Lie, and V. Cortier. Taipei, Taiwan: ACM Press, 2025, pp. 409–422. DOI: [10.1145/3719027.3744796](https://doi.org/10.1145/3719027.3744796).
- [KP23] A. Kothapalli and B. Parno. “Algebraic Reductions of Knowledge”. In: *Advances in Cryptology – CRYPTO 2023*. Ed. by H. Handschuh and A. Lysyanskaya. Cham: Springer Nature Switzerland, 2023, pp. 669–701. ISBN: 978-3-031-38551-3.
- [KP25] H. Karthikeyan and A. Polychroniadou. “OPA: One-Shot Private Aggregation with Single Client Interaction and Its Applications to Federated Learning”. In: *Advances in Cryptology – CRYPTO 2025, Part VIII*. Ed. by Y. T. Kalai and S. F. Kamara. Vol. 16007. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2025, pp. 319–353. DOI: [10.1007/978-3-032-01913-4_10](https://doi.org/10.1007/978-3-032-01913-4_10).
- [KS23] A. Kothapalli and S. Setty. *CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves*. Cryptology ePrint Archive, Paper 2023/1192. 2023. URL: <https://eprint.iacr.org/2023/1192>.
- [KS24] A. Kothapalli and S. T. V. Setty. “HyperNova: Recursive Arguments for Customizable Constraint Systems”. In: *Advances in Cryptology – CRYPTO 2024, Part X*. Ed. by L. Reyzin and D. Stebila. Vol. 14929. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2024, pp. 345–379. DOI: [10.1007/978-3-031-68403-6_11](https://doi.org/10.1007/978-3-031-68403-6_11).
- [KST22] A. Kothapalli, S. Setty, and I. Tzialla. “Nova: Recursive Zero-Knowledge Arguments from Folding Schemes”. In: *Advances in Cryptology – CRYPTO 2022, Part IV*. Ed. by Y. Dodis and T. Shrimpton. Vol. 13510. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2022, pp. 359–388. DOI: [10.1007/978-3-031-15985-5_13](https://doi.org/10.1007/978-3-031-15985-5_13).
- [KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *Advances in Cryptology – ASIACRYPT 2010*. Ed. by M. Abe. Vol. 6477. Lecture Notes in Computer Science. Singapore: Springer Berlin Heidelberg, Germany, 2010, pp. 177–194. DOI: [10.1007/978-3-642-17373-8_11](https://doi.org/10.1007/978-3-642-17373-8_11).
- [Leh+26] R. Lehmkuhl, H. Corrigan-Gibbs, E. Dauterman, and D. J. Wu. *Helix: Heavy-Light Private Aggregation*. Cryptology ePrint Archive, Paper 2026/059. 2026. URL: <https://eprint.iacr.org/2026/059>.
- [Li+23] H. Li, H. Lin, A. Polychroniadou, and S. Tessaro. “LERN: Secure Single-Server Aggregation via Key-Homomorphic Masking”. In: *Advances in Cryptology – ASIACRYPT 2023, Part I*. Ed. by J. Guo and R. Steinfeld. Vol. 14438. Lecture Notes in Computer Science. Guangzhou, China: Springer, Singapore, Singapore, 2023, pp. 302–334. DOI: [10.1007/978-981-99-8721-4_10](https://doi.org/10.1007/978-981-99-8721-4_10).
- [Lun+90] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. “Algebraic Methods for Interactive Proof Systems”. In: *31st Annual Symposium on Foundations of Computer Science*. St. Louis, MO, USA: IEEE Computer Society Press, 1990, pp. 2–10. DOI: [10.1109/FSCS.1990.89518](https://doi.org/10.1109/FSCS.1990.89518).

- [Lyc+23] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi. “RoFL: Robustness of Secure Federated Learning”. In: *2023 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2023, pp. 453–476. DOI: [10.1109/SP46215.2023.10179400](https://doi.org/10.1109/SP46215.2023.10179400).
- [Ma+23] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin. “Flamingo: Multi-Round Single-Server Secure Aggregation with Applications to Private Federated Learning”. In: *2023 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2023, pp. 477–496. DOI: [10.1109/SP46215.2023.10179434](https://doi.org/10.1109/SP46215.2023.10179434).
- [Mad+25] V. Madathil, A. Lazzaretti, Z. Liu, and C. Papamanthou. *TACITA: Threshold Aggregation without Client Interaction*. Cryptology ePrint Archive, Paper 2025/1579. 2025. URL: <https://eprint.iacr.org/2025/1579>.
- [NBS] W. D. Nguyen, D. Boneh, and S. Setty. “Revisiting the Nova Proof System on a Cycle of Curves”. In: *5th Conference on Advances in Financial Technologies (AFT 2023)*. Dagstuhl, Germany. DOI: [10.4230/LIPICs.AFT.2023.18](https://doi.org/10.4230/LIPICs.AFT.2023.18).
- [Ngu+24] W. D. Nguyen, T. Datta, B. Chen, N. Tyagi, and D. Boneh. “Mangrove: A Scalable Framework for Folding-Based SNARKs”. In: *Advances in Cryptology – CRYPTO 2024, Part X*. Ed. by L. Reyzin and D. Stebila. Vol. 14929. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2024, pp. 308–344. DOI: [10.1007/978-3-031-68403-6_10](https://doi.org/10.1007/978-3-031-68403-6_10).
- [Set20] S. Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: *Advances in Cryptology – CRYPTO 2020, Part III*. Ed. by D. Micciancio and T. Ristenpart. Vol. 12172. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2020, pp. 704–737. DOI: [10.1007/978-3-030-56877-1_25](https://doi.org/10.1007/978-3-030-56877-1_25).
- [STW24] S. T. V. Setty, J. Thaler, and R. S. Wahby. “Unlocking the Lookup Singularity with Lasso”. In: *Advances in Cryptology – EUROCRYPT 2024, Part VI*. Ed. by M. Joye and G. Leander. Vol. 14656. Lecture Notes in Computer Science. Zurich, Switzerland: Springer, Cham, Switzerland, 2024, pp. 180–209. DOI: [10.1007/978-3-031-58751-1_7](https://doi.org/10.1007/978-3-031-58751-1_7).
- [tec] H. G. M. improves suggestions with federated technology. <https://support.google.com/messages/answer/9327902>.
- [Val08] P. Valiant. “Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency”. In: *TCC 2008: 5th Theory of Cryptography Conference*. Ed. by R. Canetti. Vol. 4948. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer Berlin Heidelberg, Germany, 2008, pp. 1–18. DOI: [10.1007/978-3-540-78524-8_1](https://doi.org/10.1007/978-3-540-78524-8_1).
- [WCB25] F. Wang, S. Cohnen, and J. Bonneau. “SoK: Trusted setups for powers-of-tau strings”. In: *FC ’25: Proceedings of the 29th International Conference on Financial Cryptography and Data Security*. Miyakojima, Japan, 2025. URL: <https://eprint.iacr.org/2025/064.pdf>.
- [Xie+19] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *Advances in Cryptology – CRYPTO 2019, Part III*. Ed. by A. Boldyreva and D. Micciancio. Vol. 11694. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2019, pp. 733–764. DOI: [10.1007/978-3-030-26954-8_24](https://doi.org/10.1007/978-3-030-26954-8_24).

- [Zha+24] Z. Zhang, W. Li, Y. Guo, K. Shi, S. S. M. Chow, X. Liu, and J. Dong. “Fast RS-IOP Multivariate Polynomial Commitments and Verifiable Secret Sharing”. In: *USENIX Security 2024: 33rd USENIX Security Symposium*. Ed. by D. Balzarotti and W. Xu. Philadelphia, PA, USA: USENIX Association, 2024. URL: <https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-zongyang>.
- [Zha+25] J. Zhao, S. Setty, W. Cui, and G. Zaverucha. “MicroNova: Folding-Based Arguments with Efficient (On-Chain) Verification”. In: May 2025, pp. 1964–1982. DOI: [10.1109/SP61157.2025.00168](https://doi.org/10.1109/SP61157.2025.00168).

A Additional Preliminaries

A.1 Arguments and Commitments

We recall the standard security definitions for SNARKs and polynomial commitment schemes.

Definition A.1 (Succinct Non-Interactive Zero-Knowledge Argument of Knowledge).

A non-interactive argument $\Pi := (\text{Setup}, \text{Prove}, \text{Verify})$ for a relation $\mathcal{R}$ satisfies completeness, knowledge soundness, zero knowledge, and succinctness if the following conditions hold.

Completeness. Π satisfies completeness if for every $(x, w) \in R$,

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

Knowledge Soundness. Π satisfies knowledge soundness if for any efficient adversary $\mathcal{A}$ there exists a efficient extractor $\mathcal{E}$ such that for all randomness ρ ,

$$\Pr \left[\begin{array}{l} (x, w) \notin \mathcal{R}, \\ \text{Verify}(\text{pp}, x, \pi) = 1 \end{array} \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda), \\ (x, \pi) \leftarrow \mathcal{A}(\text{pp}, \rho), \\ w \leftarrow \mathcal{E}(\text{pp}, \rho, x, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

Zero Knowledge. Π satisfies zero knowledge if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for every efficient adversary $\mathcal{A}$ and efficient distinguisher $\mathcal{D}$,

$$\begin{aligned} & \Pr \left[\mathcal{D}(\text{pp}, x, \pi) = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda), \\ (x, w) \leftarrow \mathcal{A}(\text{pp}), \\ (x, w) \in \mathcal{R}, \\ \pi \leftarrow \text{Prove}(\text{pp}, x, w) \end{array} \right] \\ & \approx_c \Pr \left[\mathcal{D}(\text{pp}, x, \pi) = 1 \mid \begin{array}{l} (\text{pp}, \tau) \leftarrow \mathcal{S}_0(1^\lambda), \\ (x, w) \leftarrow \mathcal{A}(\text{pp}), \\ (x, w) \in \mathcal{R}, \\ \pi \leftarrow \mathcal{S}_1(\text{pp}, x, \tau) \end{array} \right]. \end{aligned}$$

Succinctness. Π satisfies succinctness if, for every instance–witness pair (x, w) , the proof size and the running time of the verifier are $\text{poly}(\lambda, |x|, \log |w|)$.

Definition A.2 (Polynomial Commitment Scheme). We say a polynomial commitment scheme $\Sigma = (\text{Setup}, \text{Com}, \text{Open}, \text{Verify})$ for message space $\mathbb{F}^{\leq k}[\vec{X}]$ satisfies completeness, extractability, zero-knowledge, and succinctness if the following conditions hold.

Completeness. Σ satisfies completeness if for any $p(\vec{X}) \in \mathbb{F}^{\leq k}[\vec{X}]$ with evaluation point $\vec{x}$,

$$\Pr \left[\text{Verify}(\text{pp}, c, \vec{x}, y, \pi) = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda, k), \\ c \leftarrow \text{Com}(\text{pp}, p(\vec{X})), \\ (y, \pi) \leftarrow \text{Open}(\text{pp}, p(\vec{X}), \vec{x}) \end{array} \right] = 1.$$

Extractability. Σ satisfies extractability if for any efficient adversaries $\mathcal{A}$, there exists an efficient extractor $\mathcal{E}$ such that for all randomness ρ ,

$$\Pr \left[\begin{array}{l} p(\vec{x}) \neq y, \\ \text{Verify}(\text{pp}, c, \vec{x}, y, \pi) = 1 \end{array} \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda, k), \\ (c, \vec{x}, y, \pi) \leftarrow \mathcal{A}(\text{pp}, \rho), \\ p(\vec{X}) \leftarrow \mathcal{E}(\text{pp}, \rho, c, \vec{x}, y, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

Zero Knowledge. Σ satisfies zero knowledge if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for every efficient adversary $\mathcal{A}$ and efficient distinguisher $\mathcal{D}$,

$$\Pr \left[\mathcal{D}(\text{pp}, c, \vec{x}, y, \pi) = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\lambda, k), \\ (p(\vec{X}), \vec{x}) \leftarrow \mathcal{A}(\text{pp}), \\ p(\vec{X}) \in \mathbb{F}^{\leq k}[\vec{X}], \vec{x} \in \mathbb{F}^\ell \\ c \leftarrow \text{Com}(\text{pp}, p(\vec{X})), \\ (y, \pi) \leftarrow \text{Open}(\text{pp}, p(\vec{X}), \vec{x}) \end{array} \right] \\ \approx_c \Pr \left[\mathcal{D}(\text{pp}, c, \vec{x}, y, \pi) = 1 \mid \begin{array}{l} (\text{pp}, \tau) \leftarrow \mathcal{S}(1^\lambda), \\ (p(\vec{X}), \vec{x}) \leftarrow \mathcal{A}(\text{pp}), \\ p(\vec{X}) \in \mathbb{F}^{\leq k}[\vec{X}], \vec{x} \in \mathbb{F}^\ell \\ (c, y, \pi) \leftarrow \mathcal{S}(\text{pp}, \vec{x}, \tau) \end{array} \right].$$

Succinctness. Σ satisfies succinctness if, for every polynomial $p(\vec{X}) \in \mathbb{F}^{\leq k}[\vec{X}]$ and evaluation point $\vec{x} \in \mathbb{F}^\ell$, the proof size and the running time of the verifier are $\text{poly}(\lambda, \ell, k)$.

A.2 Accumulation and PCD Schemes

We first define accumulation schemes and then provide additional background on PCD.

Definition A.3 (Accumulation Scheme). An accumulation scheme for a predicate $\varphi : (X, W) \rightarrow \{0, 1\}$ is a tuple of algorithms $\Pi_{\text{acc}} = (\text{Setup}_{\text{acc}}, \text{P}_{\text{acc}}, \text{V}_{\text{acc}}, \text{D}_{\text{acc}})$ such that:

- $\text{Setup}_{\text{acc}}(1^\lambda) \rightarrow \text{pp}$: given the security parameter λ , outputs public parameters pp .
- $\text{P}_{\text{acc}}(\text{acc}_1, \text{acc}_2) \rightarrow (\text{acc}, \text{pf})$: given two accumulators $\text{acc}_1 := (\text{acc}_1.x, \text{acc}_1.w)$ and $\text{acc}_2 := (\text{acc}_2.x, \text{acc}_2.w)$, outputs a new accumulator $\text{acc} := (\text{acc}.x, \text{acc}.w)$ and corrections terms pf .
- $\text{V}_{\text{acc}}(\text{acc}_1.x, \text{acc}_2.x, \text{pf}) \rightarrow \text{acc}.x$: given instances of two accumulators $\text{acc}_1.x$ and $\text{acc}_2.x$, and a corrections terms pf , outputs a new accumulator instance $\text{acc}.x$.
- $\text{D}_{\text{acc}}(\text{acc}) \rightarrow \{1, 0\}$: given an accumulator $\text{acc} := (\text{acc}.x, \text{acc}.w)$, accepts or rejects.

We assume public parameters pp are implicit inputs to P_{acc} , V_{acc} , and D_{acc} . Moreover, we say that Π_{acc} satisfies completeness and knowledge soundness if the following conditions hold.

Completeness. Let $\text{pp} \leftarrow \Pi_{\text{acc}}.\text{Setup}(1^\lambda)$ be the public parameters of the accumulation scheme. For every two satisfiable accumulators acc_1 and acc_2 such that $\text{D}_{\text{acc}}(\text{acc}_b) = 1$ for $b \in \{0, 1\}$,

$$\Pr \left[\begin{array}{l} \text{D}_{\text{acc}}(\text{acc}) = 1, \\ \text{V}_{\text{acc}}(\text{acc}_1.x, \text{acc}_2.x, \text{pf}) = \text{acc}.x \end{array} \mid (\text{acc}, \text{pf}) \leftarrow \text{P}_{\text{acc}}(\text{acc}_1, \text{acc}_2) \right] = 1.$$

Knowledge Soundness. For any efficient adversary $\mathcal{A}$, there exists an efficient extractor $\mathcal{E}$ such that for all randomness ρ ,

$$\Pr \left[\begin{array}{l} \text{D}_{\text{acc}}(\text{acc}) = 1, \\ \text{V}_{\text{acc}}(\text{acc}_1.x, \text{acc}_2.x, \text{pf}) = \text{acc}.x, \\ (D_{\text{acc}}(\text{acc}_1) \neq 1 \vee D_{\text{acc}}(\text{acc}_2) \neq 1) \end{array} \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}_{\text{acc}}(1^\lambda), \\ (\text{acc}, \text{acc}_1.x, \text{acc}_2.x, \text{pf}) \leftarrow \mathcal{A}(\text{pp}), \\ (\text{acc}_1.w, \text{acc}_2.w) \leftarrow \mathcal{E}(\text{pp}, \text{acc}, \text{acc}_1.x, \text{acc}_2.x, \text{pf}) \end{array} \right] = 1.$$

IVC. Incremental Verifiable Computation (IVC) [Val08] is a recursive argument of knowledge that enables verifiable execution of iterative computations such as $F^{(n)}$, i.e., the n -fold composition of a function F with itself. Consider a party that wishes to compute n steps starting from x_0 , and then recursively defines $x_i := F(x_{i-1}, \omega_i)$. Here x_i values are the public input, while ω_i are the secret witness. IVC recursively produces a proof certifying that x_i has been computed correctly.

At step 1, a proof Π_1 is generated for the computation $x_1 := F(x_0, \omega_1)$. At each subsequent step i , the prover supplies the previous proof Π_{i-1} , which attests to the correctness of step $i-1$, together with the witness ω_i . The circuit then verifies Π_{i-1} and computes $F(x_{i-1}, \omega_i)$, thereby producing a new proof Π_i . By definition, each proof Π_i certifies the correctness of the entire computation.

PCD. IVC is generalized by the notion of Proof-Carrying Data (PCD) [CT10], which generalizes the computation model from a straight-line sequence to directed acyclic graphs (DAGs). Consequently, PCD enables parallelism and distributed proving. Revisiting the sequential computation example $x_i = F(x_{i-1}, \omega_i)$, IVC produces proofs Π_i sequentially, so the prover time grows linearly with n . PCD can lift this to DAGs, thereby allowing parallel and distributed proof generation.

To illustrate this, consider a case in which a prover generates base proofs Π_i for computations $x_i = F(x_{i-1}, \omega_i)$. The prover then arranges these base proofs in a tree, placing the leaf nodes $((x_{i-1}, x_i), \Pi_i)_{i \in [n]}$ at the bottom. Each internal node takes two children and verifies their proofs. Recursing this procedure yields a root proof certifying the entire computation. Compared to a straight-line IVC, this approach enables higher degrees of parallelism and distributed execution.

How to build IVC/PCD? Traditional constructions of IVC/PCD relied on recursive SNARKs, where the SNARK verifier is encoded within the recursive circuit [Bit+13]. However, this approach is concretely expensive because SNARK verifier circuits are relatively inefficient, e.g., constructing IVC/PCD from Groth16 [Gro16] requires implementing pairing operations inside the circuit.

More recent works [Bün+21a; KST22; KS24; BC23] introduced the use of accumulation schemes to build IVC/PCD. These constructions achieve significantly lower recursive overhead and are far more efficient.

To provide a high-level intuition of how accumulation schemes operate, consider the sequential computation $x_{i+1} := F(x_i, \omega_i)$. At each step, the prover receives a proof (i.e., a fresh accumulator) Π_{i-1} attesting to the correctness of step $i-1$, together with a running accumulator acc_{i-1} from the previous step. Rather than verifying Π_{i-1} directly inside the circuit, the prover accumulates Π_{i-1} with acc_{i-1} to produce a new accumulator acc_i . This is done by implementing the accumulator verifier within the recursive circuit. The accumulation scheme guarantees that acc_i is valid if and only if both acc_{i-1} and Π_{i-1} are valid. Consequently, at the end of the computation, it suffices to verify only the final accumulator acc_n , which certifies the correctness of all n prior steps. The efficiency of accumulation-based IVC/PCD stems from the fact that, in practice, implementing an accumulator verifier is concretely much cheaper than implementing a full SNARK verifier, resulting in substantially smaller and more concretely efficient recursive circuits.

Sublinear IVC/PCD. An IVC/PCD scheme is called *sublinear* [Kad+25] if the proofs generated at each step, as well as their verification costs, are sublinear in the size of the step function F . As discussed above, there are two main recipes for constructing an IVC/PCD scheme: (i) building a SNARK and embedding its verifier inside the arithmetic circuit, or (ii) using accumulation schemes.

The first approach always yields a sublinear IVC/PCD scheme because the underlying argument of knowledge is succinct. However, as noted earlier, this method is inefficient.

KZH-Fold [Kad+25] formalize the notion of a *sublinear accumulation* scheme and show that an IVC/PCD construction is sublinear as long as the underlying accumulation scheme is sublinear. This stands in contrast to most state-of-the-art IVC/PCD systems such as Nova, HyperNova and

Protostar, which rely on linear-sized accumulation schemes. For example, when using Nova, even moderately sized circuits with one million constraints result in proofs as large as 80 MB [Zha+25; Kad+25], which takes several seconds to verify. Nova mitigates this issue by applying an additional succinct argument at the final step. However, generating this final SNARK is significantly more expensive than a single IVC/PCD step, e.g., $24\times$, and the resulting proof is no longer incremental.

Sublinear PCDs are particularly important for distributed environments, as in, e.g., WillowFold. In these scenarios, PCD proofs must be communicated between provers—for example, in WillowFold, the client transmits a leaf proof to the server. Concretely, in the case of WillowFold, as noted earlier, for a client circuit with approximately 1.4 million constraints, using a linear-sized PCD scheme such as Nova (without applying a final succinct argument) produces PCD proofs larger than 100 MB. Beyond Nova’s malleability and the high cost of compiling it into a zero-knowledge variant as discussed before, requiring the client to run an additional succinct argument on the leaf proof would lead to an order-of-magnitude increase in prover time, resulting in significant inefficiency.

Leaf-Decoupled PCD. Prior works predating Mangrove considered *generic* PCD topologies of arbitrary DAGs in which every node may carry a secret witness. Mangrove [Ngu+24] observed that this level of generality is often unnecessary in practice. In most real-world applications, such as WillowFold, the computation graph is a complete tree of fixed depth, where the local computations of F are performed only at the leaves, and all internal nodes merely aggregate their children.

This structural observation enables Mangrove to introduce a *decoupled* design: the proof systems and statements used at the leaves differ from those used at internal nodes. In contrast, in the plain (fully uniform) model—where all nodes are treated identically—the recursive circuit is encoded at every node, including the leaves. At the leaves, however, no accumulation of prior statements occurs; only the local computation F needs to be proven. Encoding the recursive circuit in this setting, therefore, leads to unnecessary overhead. Similarly, the internal nodes do not run F while F is still encoded in their circuit due to using the same circuit for leaves and internal nodes.

Mangrove avoids this inefficiency by allowing leaf nodes to prove only the local computation F , while internal nodes use specialized circuits whose sole purpose is to accumulate their children’s proofs. Since internal nodes do not perform local computation, they do not need to encode the function F . This separation significantly reduces the circuit size at both the leaves and the internal nodes, leading to more efficient recursive proofs overall. This decoupling is reflected in our PCD definition by employing two different proving functionalities for the leaf nodes and internal nodes.

Below, we recall the formal definition of leaf-decoupled PCD. Before doing so, we define a the PCD transcript and the notion of complaint PCD transcript.

Definition A.4 (Proof-Carrying Transcript). *A transcript PCDT is a DAG where each vertex $u \in \text{PCDT.Nodes}$ is labeled with local data $w^{(u)} \in \{0, 1\}^* \cup \{\perp\}$, and each edge $e \in \text{PCDT.Edges}$ is labeled with a message–proof pair $(z^{(e)}, \pi^{(e)})$. The output of PCDT, denoted PCDT.Output , is the message z corresponding to the lexicographically first sink of PCDT. Let m be the fan-in of PCDT, i.e., the maximum number of incoming edges for any vertex $v \in \text{PCDT.Nodes}$. We consider classes of compliance predicates $\mathcal{F}$ where every $\varphi \in \mathcal{F}$ is of the form*

$$\varphi : \{0, 1\}^* \times (\{0, 1\}^* \cup \{\perp\})^{m+1} \rightarrow \{0, 1\},$$

that takes as input $(z, w, z^{(1)}, \dots, z^{(m_0)}, \perp, \dots, \perp)$ for $m_0 \leq m$ and outputs accept or reject.

Proof-Carrying Transcript for Tree. A special case of a directed acyclic graph (DAG) is a *decoupling tree*. In a decoupling tree, all computation takes place at the leaves, and therefore only the leaves carry witness, while internal nodes are witness-free and only aggregate the child proofs.

Consequently, only the leaf vertices are labeled with local data w . For a leaf node, the input to the predicate has the form $(z, w, \perp, \perp, \dots, \perp)$, where w denotes the local witness associated with that leaf. In contrast, internal nodes do not have a local witness. Their predicate input consists of the node's public input z together with the outputs of their children. More precisely, for an internal node with $m_0 \leq m$ children, the input to the predicate is $(z, \perp, z_1, \dots, z_{m_0}, \perp, \dots, \perp)$.

Definition A.5 (Leaf-Decoupled Proof Carrying-Data Scheme). Fix a message space $\mathcal{Z}$ with a predicate $\text{isBase} : \mathcal{Z} \rightarrow \{0, 1\}$ and a leaf data space $\mathcal{L}$. A leaf-decoupled PCD for a class of compliance predicates $\mathcal{F}$ is a tuple of algorithms $\Pi_{\text{PCD}} := (\text{Setup}, \text{P}_{\text{leaf}}, \text{P}_{\text{internal}}, \text{V}_{\text{leaf}}, \text{V}_{\text{internal}})$ where:

- $\text{Setup}(1^\lambda) \rightarrow \text{pp}$: given the security parameter 1^λ , outputs public parameters pp .
- $\text{P}_{\text{leaf}}(z, w) \rightarrow \pi$: given a message $z \in \mathcal{Z}$, local data $w \in \mathcal{L}$, outputs a proof π .
- $\text{P}_{\text{internal}}(z, [(z^{(i)}, \pi^{(i)})]_{i=1}^k) \rightarrow \pi$: given z and message-proofs $[(z^{(i)}, \pi^{(i)})]_{i=1}^k$, outputs a proof π .
- $\text{V}_{\text{leaf}}(z, \pi) \rightarrow \{0, 1\}$: given $z \in \mathcal{Z}$ and a proof π , accepts or rejects.
- $\text{V}_{\text{internal}}(z, \pi) \rightarrow \{0, 1\}$: given $z \in \mathcal{Z}$, and a proof π , accepts or rejects.

Public parameters pp are implicit inputs to other algorithms. We require that $\text{P}_{\text{internal}}$ reduces the proof size and that the resulting proof size is independent of the number of nodes.

Completeness. Let $\text{pp} \in \text{Setup}(1^\lambda)$ be the public parameters and $\varphi \in \mathcal{F}$ be the predicate.

- *Leaf Completeness.* If $\varphi(z, w) = 1$ and $\pi \leftarrow \text{P}_{\text{leaf}}(z, w)$, then it holds that: $\text{V}_{\text{leaf}}(z, \pi) = 1$.
- *Internal Completeness.* For all $1 \leq i \leq k$, suppose that

$$\begin{cases} \text{V}_{\text{leaf}}(z^{(i)}, \pi^{(i)}) = 1, & \text{if } \text{isBase}(z^{(i)}) = 1, \\ \text{V}_{\text{internal}}(z^{(i)}, \pi^{(i)}) = 1, & \text{if } \text{isBase}(z^{(i)}) = 0. \end{cases}$$

Then, for any proof $\pi \leftarrow \text{P}_{\text{internal}}(z, [(z^{(i)}, \pi^{(i)})]_{i=1}^k)$, it holds that: $\text{V}_{\text{internal}}(z, \pi) = 1$.

Knowledge Soundness. Similarly, it is defined for the leaf and internal nodes as below:

- *Leaf Knowledge Soundness.* We say that Π_{PCD} satisfies leaf knowledge soundness if $(\text{Setup}, \text{P}_{\text{leaf}}, \text{V}_{\text{leaf}})$ is an argument of knowledge for the verification predicate of V_{leaf} .
- *Internal Node Knowledge Soundness.* Consider the relations

$$\mathcal{R}_1 = \left\{ ((z^{(i)})_{i=1}^k, (\pi^{(i)})_{i=1}^k) \mid \begin{cases} \text{V}_{\text{leaf}}(z^{(i)}, \pi^{(i)}) = 1, & \text{if } \text{isBase}(z^{(i)}) = 1, \\ \text{V}_{\text{internal}}(z^{(i)}, \pi^{(i)}) = 1, & \text{if } \text{isBase}(z^{(i)}) = 0, \end{cases} \right\},$$

and $\mathcal{R}_2 = \{(\varphi, z, \pi) \mid \text{V}_{\text{internal}}(z, \pi) = 1\}$. We say that Π_{PCD} satisfies internal node knowledge soundness if $(\text{Setup}, \text{P}_{\text{internal}}, \text{V}_{\text{internal}})$ is a reduction of knowledge [KP23] from $\mathcal{R}_1$ to $\mathcal{R}_2$.

Zero Knowledge. We say that Π_{PCD} satisfies leaf zero knowledge if the tuple $(\text{Setup}, \text{P}_{\text{leaf}}, \text{V}_{\text{leaf}})$ is a zero-knowledge argument for the verification predicate of V_{leaf} .

A.3 Deferred Details of KZH and KZH-Fold

In this section, we present the details of KZH-k and KZH-Fold-k, along with the associated notation.

Notation. We use the same notation as in [Kad+25]. We denote the space of n dimensional tensors with dimensions $(d_1, d_2, \dots, d_n)$ on field $\mathbb{F}$ with $\mathbb{F}^{d_1 \times d_2 \times \dots \times d_n}$. For $\mathbf{T} \in \mathbb{F}^{d_1 \times d_2 \times \dots \times d_k}$ and $\vec{x}_1 \in \mathbb{F}^{d_1}$, we denote tensor inner product as

$$\langle \mathbf{T}, \vec{x}_1 \rangle = \langle \mathbf{T}_1, \vec{x}_1 \rangle \otimes \mathbf{T}_2 \otimes \dots \otimes \mathbf{T}_k.$$

Note that the result is a tensor in $\mathbb{F}^{d_2 \times \dots \times d_k}$. Similarly for $\vec{x}_1 \in \mathbb{F}^{d_1}$ and $\vec{x}_2 \in \mathbb{F}^{d_2}$, it holds that $\langle \mathbf{T}, \vec{x}_1 \otimes \vec{x}_2 \rangle = \langle \langle \mathbf{T}, \vec{x}_1 \rangle, \vec{x}_2 \rangle \in \mathbb{F}^{d_3 \times \dots \times d_k}$.

KZH Protocol. KZH- k is a commitment scheme for such tensors. In particular, $\mathbf{T}$ represents the polynomial that can be opened at evaluation point $(\vec{x}_1, \dots, \vec{x}_k)$ for $\vec{x}_j \in \mathbb{F}^{d_j}$ as:

$$\langle \langle \langle \mathbf{T}, \vec{x}_1 \rangle, \vec{x}_2 \rangle, \dots, \vec{x}_{k-1} \rangle, \vec{x}_k \rangle = y \in \mathbb{F}.$$

The scheme admits an efficient accumulation scheme and supports both multilinear and univariate polynomial commitments. The core insight is that we commit to $\text{com} = \langle \mathbf{T}, \vec{\mu}_1 \otimes \vec{\mu}_2 \otimes \dots \otimes \vec{\mu}_k \rangle \times \mathbb{G}$, for secret elements $\vec{\mu}_1, \dots, \vec{\mu}_k$ and group generator $\mathbb{G}$, and open $[\text{com}_i]_{i=1}^k$, commitments to all $k-1$ dimensional slices of $\mathbf{T}$. The verifier can check that these slices are correct using a pairing with $\vec{\mu}_1$ and compute $\text{com}' = \langle (\text{com}_1, \dots, \text{com}_k), \vec{x}_1 \rangle$. com' is now a commitment to the $k-1$ dimensional tensor $\mathbf{T}_1 = \langle \mathbf{T}, \vec{x}_1 \rangle$ and we can recursively apply the scheme. We present the full protocol for degree $[d]^k := (d, d, \dots, d)$ tensor in Figure 8. IronDict [Haf+25, Appendix E] describes the KZH- k construction for degree $(d_1, \dots, d_k)$, but we omit it here and since do not use it.

KZH-Fold protocol description. KZH-Fold an accumulation scheme for KZH- k commitment scheme with accumulator size and the decider time are $O(k \cdot n^{\frac{1}{k}})$. The accumulation verifier performs $O(k)$ $\mathbb{G}_1$ operations. For a kd -multilinear polynomial, the accumulator instance and witness can be described as follows: (red terms only appear in accumulators, not proofs)

$$\text{acc}.x = \{\text{com}, \text{com}_1, \dots, \text{com}_{k-1} \in \mathbb{G}, \vec{x}_1, \dots, \vec{x}_k \in \mathbb{F}^d, y \in \mathbb{F}, \textcolor{red}{E}_{\mathbb{G}} \in \mathbb{G}, \textcolor{red}{e}_{\mathbb{F}} \in \mathbb{F}\}$$

$$\text{acc}.w = \{[D_{i,j}]_{j \in [k-1], i \in [0,d]}, \mathbf{T}_k \in \mathbb{F}^d\}$$

The accumulation prover, verifier and decider are defined in Figures 9 and 10.

B Zero-Knowledge KZH

Zero-Knowledge KZH. Prior work [Bün+21b] proposes a compiler that transforms a homomorphic PCS to one that has a zero-knowledge opening protocol. IronDict [Haf+25] shows how to use this compiler for KZH while only using a sublinear amount of randomness. To give an overview, let $f(\vec{X})$ be an ℓ -variate multilinear polynomial, and let $h \in \mathbb{G}$ be a generator. We commit to $f(\vec{X})$ as $C_{\text{hide}} := \text{KZH.Com}(f) + \tau \times h$ where $\tau \in \mathbb{F}$ is the randomness. To open $f(\vec{X})$ at an evaluation point $(\vec{x}, y)$, the prover samples a random ℓ -variate multilinear polynomial $r(\vec{X})$ and sends $y_R := r(\vec{x})$ and $R_{\text{hide}} := \text{KZH.Com}(r) + \rho \times h$ to the verifier where ρ is the randomness. Upon a challenge α , the prover sends $\alpha \cdot \tau + \rho$ to the verifier, and the verifier sets $C_{\text{lin}} := \alpha \times C_{\text{hide}} + R_{\text{hide}} - (\alpha \cdot \tau + \rho) \times h$. Finally, the prover opens C_{lin} at $\vec{x}$ which proves to be evaluated to $\alpha \cdot y + y_r$. Formal description is provided in Figure 11. IronDict shows that for KZH- k it suffices to sample $r(\vec{X})$ using $O(k \cdot N^{1/k})$ number of non-zero evaluation points, where N is the size of $f(\vec{X})$. This means that making a KZH- k opening zero-knowledge requires only $O(k \cdot N^{1/k})$ extra group operations.

KZH- k .Setup:

1. Sample $\mathbf{G} \leftarrow \mathbb{G}_1, \mathbf{V} \leftarrow \mathbb{G}_2$ and $[[\mu_{i,j}]_{i=1}^d]_{j=1}^k \leftarrow \mathbb{F}$
2. $\mathbf{H}_1 = \{\mathbf{H}_{i_1, \dots, i_k} \leftarrow (\prod_{j=1}^k \mu_{i_j, j}) \times \mathbf{G} : \forall i_1, \dots, i_k \in [d]\}$
3. $\mathbf{H}_2 = \{\mathbf{H}_{i_2, \dots, i_k} \leftarrow (\prod_{j=2}^k \mu_{i_j, j}) \times \mathbf{G} : \forall i_2, \dots, i_k \in [d]\}$
- ...
4. $\mathbf{H}_k = \{\mathbf{H}_{i_k} \leftarrow \mu_{i_k, k} \times \mathbf{G} : \forall i_k \in [d]\}$
5. $\mathbf{V} = \{\mathbf{V}_{i,j} \leftarrow \mu_{i,j} \times \mathbf{V} : \forall i \in [d], j \in [k]\}$
6. Output $(\mathbf{G}, \mathbf{V}, \mathbf{H}_1, \mathbf{H}_2, \dots, \mathbf{H}_k, \mathbf{V})$

KZH- k .Com: For $\mathbf{T} \in \mathbb{F}^{d \times d \times \dots \times d}$, compute the commitment as it follows:

1. Output $\text{com} \leftarrow \langle \mathbf{T}, \mathbf{H}_1 \rangle = \sum_{(i_1, i_2, \dots, i_k) \in [d]^k} \mathbf{T}_{i_1, i_2, \dots, i_k} \times \mathbf{H}_{i_1, i_2, \dots, i_k}$

KZH- k .Open: Given commitment $\text{com} \in \mathbb{G}_1$, tensor $\mathbf{T} \in \mathbb{F}^{d \times d \times \dots \times d}$ and inputs $\vec{x}_j \in \mathbb{F}^d$:

1. Let $\mathbf{T}_1 = \mathbf{T}$
2. For $j = 1, \dots, k-1$:
 - (a) Set $\mathbf{T}_{j+1} \leftarrow \langle \mathbf{T}_j, \vec{x}_j \rangle$
 - (b) Compute vector $\vec{D}_j$ e.g. $D_j[i] \leftarrow \langle \langle i \rangle \otimes \mathbf{T}_j, \mathbf{H}_{j+1} \rangle$ for all $i \in [d]$.
3. Output $\pi \leftarrow \{[\vec{D}_j]_{j \in [k-1]}, \mathbf{T}_k\}$

KZH- k .Verify:] Given commitment $\text{com} \in \mathbb{G}_1$, inputs $\vec{x}_j \in \mathbb{F}^d$, output $y \in \mathbb{F}$ and opening proof π :

1. Parse $\{[\vec{D}_j]_{j \in [k-1]}, \mathbf{T}_k\} \leftarrow \pi$
2. Set $\text{com}_0 = \text{com}$
3. For $j \in 1, \dots, k-1$
 - (a) Check that $e(\text{com}_{j-1}, \mathbf{V}) = \sum_{i=0}^d e(D_j[i], \mathbf{V}_{i,j})$
 - (b) Compute $\text{com}_j \leftarrow \langle \vec{x}_j, \vec{D}_j \rangle$
4. Check that $\text{com}_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k \rangle$
5. Check that $\langle \mathbf{T}_k, \vec{x}_k \rangle = y$

Figure 8: KZH-k description.

$P_{\text{acc}}((\text{acc}.x, \text{acc}.w), (\text{acc}'.x, \text{acc}'.w)):$

1. Parse $\text{acc}.x, \text{acc}.w$ and $\text{acc}'.x, \text{acc}'.w$ as below:

- (a) $\{\text{com}, \text{com}_1, \dots, \text{com}_{k-1}, (\vec{x}_1, \dots, \vec{x}_k, y), (E_{\mathbb{G}}, e_{\mathbb{F}})\} \leftarrow \text{acc}.x$
- (b) $\{\{[D_{i,j}]_{j \in [k-1], i \in [0,d]}, \mathbf{T}_k\} \leftarrow \text{acc}.w$
- (c) $\{\text{com}', \text{com}'_1, \dots, \text{com}'_{k-1}, (\vec{x}'_1, \dots, \vec{x}'_k, y'), (E'_{\mathbb{G}}, e'_{\mathbb{F}})\} \leftarrow \text{acc}'.x$
- (d) $\{\{[D'_{i,j}]_{j \in [k-1], i \in [0,d]}, \mathbf{T}'_k\} \leftarrow \text{acc}'.w$

2. Let $\text{Dec}_{\mathbb{G}}$ and $\text{Dec}_{\mathbb{F}}$ be the verification checks as defined in Figure 10. Compute $Q \in \mathbb{G}$ such that:

$$\begin{aligned} \text{Dec}_{\mathbb{G}}((1-X) \cdot (\vec{x}_j, \text{com}_j, [D_{i,j}]_{i=0}^d) + X \cdot (\vec{x}'_j, \text{com}'_j, [D'_{i,j}]_{i=0}^{d_j})) \\ = (1-X) \times E_j + X \times E'_j + (1-X) \cdot X \times Q. \end{aligned}$$

and $q \in \mathbb{F}$ such that

$$\begin{aligned} \text{Dec}_{\mathbb{F}}((1-X) \cdot (\mathbf{T}_k, \vec{x}_k, y) + X \cdot (\mathbf{T}'_k, \vec{x}'_k, y')) \\ = (1-X) \cdot e_{\mathbb{F}} + X \cdot e'_{\mathbb{F}} + (1-X) \cdot X \cdot q. \end{aligned}$$

3. Derive challenge $\alpha \leftarrow H(\text{acc}.x, \text{acc}'.x, Q, q)$ through Fiat-Shamir.

4. Compute

- (a) $\text{com}'' \leftarrow (1-\alpha) \times \text{com} + \alpha \times \text{com}'$
- (b) $\vec{x}_i'' \leftarrow (1-\alpha) \cdot \vec{x}_i + \alpha \cdot \vec{x}_i'$ for $i \in [k]$
- (c) $y'' \leftarrow (1-\alpha) \cdot y + \alpha \cdot y'$
- (d) $\text{com}''_j \leftarrow (1-\alpha) \times \text{com}_j + \alpha \times \text{com}'_j$ for $j \in [1, k-1]$
- (e) $D''_{i,j} \leftarrow (1-\alpha) \times D_{i,j} + \alpha \times D'_{i,j}$ for each j, i for $i \in [k]$ and $j \in [k-1]$
- (f) $E''_{\mathbb{G}} \leftarrow (1-\alpha) \times E_{\mathbb{G}} + \alpha \times E'_{\mathbb{G}} + (1-\alpha) \cdot \alpha \times Q$
- (g) $e''_{\mathbb{F}} \leftarrow (1-\alpha) \cdot e_{\mathbb{F}} + \alpha \cdot e'_{\mathbb{F}} + (1-\alpha) \cdot \alpha \cdot q$
- (h) $\mathbf{T}''_k \leftarrow (1-\alpha) \times \mathbf{T}_k + \alpha \times \mathbf{T}'_k$

5. Output the new accumulator $\text{acc}'' = (\text{acc}'' .x, \text{acc}'' .w)$ and $\text{pf} = \{Q, q\}$ as the accumulation proof.

$$\begin{aligned} \text{acc}'' .x &= \{\text{com}'', [\text{com}''_j]_{j \in [k-1]}, [\vec{x}_i'']_{i \in [k]}, y'', [E''_{\mathbb{G}}, e_{\mathbb{F}}]\} \\ \text{acc}'' .w &= \{\{[D''_{i,j}]_{j \in [k-1], i \in [0,d_j]}, \mathbf{T}''_k\}\} \end{aligned}$$

Figure 9: KZH-k accumulation prover.

$V_{\text{acc}}(\text{acc}.x, \text{acc}'.x, \text{pf} = \{Q \in \mathbb{G}, q \in \mathbb{F}\})$:

1. Input $\text{acc}.x, \text{acc}'.x$ and $\text{pf} = \{Q, q\}$.
2. Regenerate challenge $\alpha \leftarrow H(A.X, A'.X, \text{pf})$
3. Compute
 - (a) $\text{com}'' \leftarrow (1 - \alpha) \times \text{com} + \alpha \times \text{com}'$
 - (b) $\vec{x}_i'' \leftarrow (1 - \alpha) \cdot \vec{x}_i + \alpha \cdot \vec{x}_i'$ for $i \in [k]$
 - (c) $y'' \leftarrow (1 - \alpha) \cdot y + \alpha \cdot y'$
 - (d) $\text{com}_j'' \leftarrow (1 - \alpha) \times \text{com}_j + \alpha \times \text{com}_j'$ for $j \in [1, k - 1]$
 - (e) $E_{\mathbb{G}}'' \leftarrow (1 - \alpha) \times E_{\mathbb{G}} + \alpha \times E'_{\mathbb{G}} + (1 - \alpha) \cdot \alpha \times Q$
 - (f) $e_{\mathbb{F}}'' \leftarrow (1 - \alpha) \cdot e_{\mathbb{F}} + \alpha \cdot e'_{\mathbb{F}} + (1 - \alpha) \cdot \alpha \times q$
4. Output the new accumulator instance $\text{acc}'' .x$.

$$\text{acc}'' .x = \{\text{com}'', [\text{com}_j'']_{j \in [k-1]}, [\vec{x}_i'']_{i \in [k]}, y'', E_{\mathbb{G}}'', e_{\mathbb{F}}''\}$$

$D_{\text{acc}}(\text{acc}.x, \text{acc}.w)$:

1. Parse instance and witness as follows:
 - (a) $\{\text{com}, (\text{com}_1, \dots, \text{com}_{k-1}), (\vec{x}_1, \dots, \vec{x}_k, y), E_{\mathbb{G}}, e_{\mathbb{F}}\} \leftarrow \text{acc}.x$
 - (b) $\{[\vec{D}_j = [\text{D}_{i,j}]_{i \in [0,d]}]_{j=1}^{k-1}, \mathbf{T}_k\} \leftarrow \text{acc}.w$
2. Set $\text{com}_0 \leftarrow \text{com}$
3. For each $j \in [1, k - 1]$ check that

$$e(\text{com}_{j-1}, \mathbf{V}) - \sum_{i=0}^d e(\text{D}_{i,j}, \mathbf{V}_{i,j}) = 0$$

4. Check that $\text{Dec}_{\mathbb{G}}([\vec{x}_i]_{i=1}^k \text{com}, [\text{com}_i]_{i=1}^{k-1}, [\vec{D}_j \in \mathbb{G}^d]_{j=1}^{k-1}) = \sum_{j=1}^{k-1} (\text{com}_j - \langle \vec{x}_j, \vec{D}_j \rangle) = E_{\mathbb{G}}$
5. Check that $\langle \mathbf{T}_k, \mathbf{H}_k \rangle = \text{com}_{k-1}$
6. Check that $\text{Dec}_{\mathbb{F}}(\vec{x}_k, \mathbf{T}_k, y) = \langle \mathbf{T}_k, \vec{x}_k \rangle - y = e_{\mathbb{F}}$

Figure 10: KZH-k accumulation verifier and decider.

zkSetup : Run $\text{KZH.Setup}(1^\lambda, k)$ and also outputs a new generator $h \in \mathbb{G}_1$.

zkCommit :

1. The prover samples randomness $\tau \leftarrow \mathbb{F}$
2. Then computes the commitment $C \leftarrow \text{KZH.Commit}(f(\vec{X}))$, and outputs

$$C_{\text{hide}} := C + \tau \cdot h$$

zkOpen :

1. The prover samples $r(\vec{X})$ which has the same size as $f(\vec{X})$ and samples randomness $\rho \leftarrow \mathbb{F}$.
2. The prover runs $R_{\text{hide}} \leftarrow \text{zkCommit}_{\text{KZH-k}}(r(\vec{X}))$ with randomness ρ .
3. The prover computes $y_R \leftarrow r(\vec{x})$ and then sends (R_{hide}, y_R) to the verifier.
4. The prover receives challenge α and computes $\rho' \leftarrow \alpha \cdot \tau + \rho$ and sends it to the verifier.
5. The prover computes $C_{\text{lin}} := \alpha \times C_{\text{hide}} + R_{\text{hide}} - \rho' \times h$.
6. The prover runs $\pi \leftarrow \text{KZH.Open}(\alpha \times f(\vec{X}) + r(\vec{X}), C_{\text{lin}}, \vec{x})$ and outputs proof π .

zkVerify :

1. The verifier receives (R_{hide}, y_R) from the prover and sends challenge α and receives ρ' .
2. The verifier computes $C_{\text{lin}} := \alpha \times C_{\text{hide}} + R_{\text{hide}} - \rho' \times h$.
3. The verifier computes $y_{\text{lin}} := \alpha \cdot y + y_R$.
4. The verifier runs $\text{Verify}_{\text{KZH-k}}(C_{\text{lin}}, \vec{x}, y_{\text{lin}}, \pi)$ and outputs its result.

Figure 11: The zero-knowledge compiler for KZH-k .

Accumulation (Folding) of Zero-Knowledge KZH-k . A notable feature of the zero-knowledge compiler for the KZH commitments is that it does not alter the underlying verification algorithm. The resulting verifier performs the standard KZH verification and also: (1) derives a challenge by hashing the pair (R_{hide}, y_R) ; (2) performs a 3 number of group operations to compute C_{lin} ; and (3) computes y_{lin} using field operations. In practice, we use KZH-Fold to accumulate the KZH verifier, while the remaining operations are encoded inside the IVC/PCD step circuit. The most expensive step is the group computation required for C_{lin} . However, the KZH-Fold verifier already has $k + 2$ group operations (for KZH-k), so adding just three additional operations remains acceptable. In concrete instantiations, these group operations can be handled efficiently using the CycleFold compiler [KS23]. We omit the details here.

Game $\text{Sim-Ext}_{0,\Pi}^{\text{Sim},\mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}, \text{Sim}}(\text{pp})$ • $b \leftarrow \Pi.\text{Verify}^{\mathcal{H}}(\text{pp}, x, \pi)$ • Return $b \wedge ((x, \pi) \notin Q_{\text{Sim}})$	Game $\text{Sim-Ext}_{1,\Pi,R}^{\mathcal{E}, \text{Sim}, \mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}, \text{Sim}}(\text{pp})$ • $b \leftarrow \Pi.\text{Verify}^{\mathcal{H}}(\text{pp}, x, \pi)$ • $w \leftarrow \mathcal{E}^{\mathcal{A}}(\text{pp}, x, \pi, Q_{\mathcal{O}}, Q_{\mathcal{H}}, Q_{\text{Sim}})$ • Return $b \wedge ((x, \pi) \notin Q_{\text{Sim}}) \wedge ((x, w) \in R)$
---	---

Figure 12: The simulation extractability security games. $\mathcal{H}$ denotes the random oracle, $\mathcal{O}$ denotes the srs update oracle, and Sim is a simulator oracle for Π . Sim takes as input x and outputs a proof π and may reprogram $\mathcal{H}$. $Q_{\mathcal{O}}$, $Q_{\mathcal{H}}$, and Q_{Sim} record queries to $\mathcal{O}$, $\mathcal{H}$, and Sim oracles, respectively. The extractor $\mathcal{E}$ is given black-box access to $\mathcal{A}$, can rewind $\mathcal{A}$, and implements $\mathcal{H}$ and Sim for $\mathcal{A}$.

C Simulation Extractability

In this section, we prove our instantiation of PCD satisfies leaf-SE. It suffices for us to prove that the base leaf proofs, which are instances of zk-Spartan+zk-KZH, satisfy simulation extractability.

C.1 Simulation Extractability of zk-Spartan+zk-KZH

Below, we first recall the definition of simulation extractability for non-interactive arguments.

Definition C.1 (Simulation Extractability). *Let $\Pi := \langle \text{Setup}, \text{Prove}, \text{Verify} \rangle$ be a non-interactive zero-knowledge argument, and Sim be a zero-knowledge simulator for Π . Then, we say that Π is simulation extractable with respect to the simulator Sim if there exists an efficient extractor $\mathcal{E}$ such that for every efficient adversary $\mathcal{A}$,*

$$\text{Sim-Ext}_{0,\Pi}^{\text{Sim},\mathcal{A}}(1^\lambda) \approx_c \text{Sim-Ext}_{1,\Pi,R}^{\mathcal{E}, \text{Sim}, \mathcal{A}}(1^\lambda).$$

Below, we recall the definitions of special soundness, trapdoor-less zero-knowledge, and unique response—three properties that we require for the argument zk-Spartan+zk-KZH to prove its SE. To define special soundness, we first need to introduce the notion of a tree of accepting transcripts.

Definition C.2 (Tree of Accepting Transcripts). *An $(n_1, \dots, n_r)$ -tree of accepting transcripts is a tree in which each node at depth $i \in [r+1]$ is the i th prover message of an accepting transcript. Edges are labeled by challenges such that no two edges at the same depth share the same label. Each node at depth i has $n_i - 1$ siblings and n_{i+1} children. The total number of branches is $N = \prod_i n_i$.*

Definition C.3 (Special Soundness). *Let $\Pi := \langle \text{Setup}, \text{Prove}, \text{Verify} \rangle$ denote a $(2r+1)$ -message and FS-transformed non-interactive zero-knowledge argument. For any $\vec{n} = (n_1, \dots, n_r) \in \mathbb{N}^r$, We say that Π satisfies $\vec{n}$ -special soundness if there exists an efficient tree-extraction algorithm $\mathcal{TE}$, such that for every efficient adversary $\mathcal{A}$,*

$$\text{Special}_{0,\Pi}^{\mathcal{A}}(1^\lambda) \approx_c \text{Special}_{1,\Pi,\vec{n}}^{\mathcal{TE}, \mathcal{A}}(1^\lambda).$$

Definition C.4 (k -Trapdoor-Less Zero Knowledge). *Assume that $\Pi := \langle \text{Setup}, \text{Prove}, \text{Verify} \rangle$ is a $(2r+1)$ -message and FS-transformed non-interactive zero-knowledge argument. We say that Π*

Game $\text{Special}_{0,\Pi}^{\mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}}(\text{pp})$ • $b \leftarrow \Pi.\text{Verify}^{\mathcal{H}}(\text{pp}, x, \pi)$ • Return b	Game $\text{Special}_{1,\Pi,\vec{n}}^{\mathcal{TE}, \mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}}(\text{pp})$ • $b_0 \leftarrow \Pi.\text{Verify}^{\mathcal{H}}(\text{pp}, x, \pi)$ • $\mathcal{T}, w \leftarrow \mathcal{TE}^{\mathcal{A}}(\text{pp}, x, \pi, Q_{\mathcal{O}}, Q_{\mathcal{H}})$ • $b_1 \leftarrow \text{IsAccepting}^{\mathcal{H}}(\text{pp}, x, \vec{n}, \mathcal{T})$ • Return $b_0 \wedge b_1 \wedge ((x, w) \in R)$
--	---

Figure 13: The special soundness security games. IsAccepting is an efficient algorithm that checks whether $\mathcal{T}$ is an $\vec{n}$ -tree of accepting transcripts with respect to the argument Π and instance x .

Game $k\text{-TLZK}_{0,\Pi}^{\text{P}, \mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $b \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}, \text{P}}(\text{pp})$ • Return b <u>$\text{P}(\text{pp}, x, w)$</u> • If $(x, w) \notin R$ abort • Else return $\Pi.\text{Prove}(\text{pp}, x, w)$	Game $k\text{-TLZK}_{1,\Pi}^{\text{Sim}, \mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $b \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}, \text{Sim}}(\text{pp})$ • Return b <u>$\text{Sim}(\text{pp}, x, w)$</u> • If $(x, w) \notin R$ abort • Else return $\text{Sim}^{k\text{-TLZK}}(\text{pp}, x)$
--	---

Figure 14: The k -trapdoor-less zero-knowledge security games. $\text{Sim}^{k\text{-TLZK}}$ is a zero-knowledge simulator for Π such that for each query (x, π) , it reprograms $\mathcal{H}$ only once in round k .

satisfies k -trapdoor-less zero-knowledge for a $k \in [1, r]$, if there exists a efficient simulator $\text{Sim}^{k\text{-TLZK}}$ such that for every efficient adversary $\mathcal{A}$,

$$k\text{-TLZK}_{0,\Pi}^{\text{P}, \mathcal{A}}(1^\lambda) \approx k\text{-TLZK}_{1,\Pi}^{\text{Sim}, \mathcal{A}}(1^\lambda).$$

Definition C.5 (k -Unique Response). Let $\Pi := \langle \text{Setup}, \text{Prove}, \text{Verify} \rangle$ be a $(2r + 1)$ -message and FS-transformed non-interactive zero-knowledge argument. Then, we say that Π satisfies k -unique response for a $k \in [1, r]$, if for every efficient adversary $\mathcal{A}$,

$$\Pr[k\text{-UR}_{\Pi}^{\mathcal{A}}(1^\lambda)] \leq \text{negl}(\lambda).$$

The following lemma shows that simulation extractability of a FS-transformed non-interactive argument follows from special soundness, trapdoor-less zero-knowledge, and unique response.

Lemma C.1. [Gan+22] Let $\Pi := \langle \text{Setup}, \text{Prove}, \text{Verify} \rangle$ be a $(2r + 1)$ -message and FS-transformed non-interactive zero-knowledge argument. Then, Π is simulation extractable if it has (updatable-SRS) special soundness, k -unique response, and k -trapdoor-less zero knowledge for the same $k \in [1, r]$.

Game $k\text{-UR}_{\Pi}^A(1^\lambda)$

- $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$
- $(x, \pi_0, \pi_1, c) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{H}}(\text{pp})$
- $b_0 \leftarrow \Pi.\text{Verify}^{\mathcal{H}[(\text{pp}, x, \pi_0|_k) \mapsto c]}(\text{pp}, x, \pi_0)$
- $b_1 \leftarrow \Pi.\text{Verify}^{\mathcal{H}[(\text{pp}, x, \pi_1|_k) \mapsto c]}(\text{pp}, x, \pi_1)$
- **Return** $b_0 \wedge b_1 \wedge (\pi_0 \neq \pi_1) \wedge (\pi_0|_k = \pi_1|_k)$

Figure 15: The k -unique response security game. $\mathcal{H}[(\text{pp}, x, \pi_b|_k) \mapsto c]$ indicates that $\mathcal{H}$ has been reprogrammed to c on the input $(\text{pp}, x, \pi_b|_k)$, where $\pi_b|_k$ denotes proof messages for rounds 1 to k .

The special soundness of zk-Spartan+zk-KZH follows from the knowledge soundness of KZH polynomial commitments and Spartan PIOP. Below, we revisit the proof of knowledge soundness for the KZH polynomial commitment and adapt it to the updatable-SRS setting—Kadianakis et al. [Kad+25, Theorem 3] proved knowledge soundness only in the standalone setting where the adversary receives one SRS and does not call the SRS update oracle.

Lemma C.2. *KZH satisfies knowledge soundness in the updatable-SRS setting.*

Proof. Kadianakis et al. [Kad+25, Theorem 3] showed that KZH- k is straight-line extractable in the AGM under the DLog and CDH assumptions. Here, we restate their proof in the updatable-SRS setting, where the adversary may also query the SRS update oracle $\mathcal{O}$ arbitrarily many times.

Let s denote the number of SRS updates, and $\text{srs}^{(u)} = (G, V, \{\mathbf{H}_r^{(u)}\}_{r \in [k]}, \mathbf{V}^{(u)})$ be the SRS after the u th update, where $\text{srs}^{(0)}$ denotes the initial SRS. For each $u \in [s]$ and $r \in [k]$, we have

$$\mathbf{V}_{*,r}^{(u)} := \alpha_r^{(u)} \cdot \mathbf{V}_{*,r}^{(u-1)} \quad \text{and} \quad \mathbf{H}_r^{(u)} = \alpha_r^{(u)} \cdot \mathbf{H}_r^{(u-1)},$$

where $\alpha_r^{(u)}$ is the update randomness sampled by the update oracle $\mathcal{O}$ on the u th update query.

Let $\mathcal{A}$ be a PPT adversary that, on input $\{\text{srs}^{(u)}\}_{u \in [s]}$, outputs a commitment C and a tuple $((x_1, \dots, x_k), y, \pi)$ that is accepted by the verifier. We construct an extractor $\mathcal{E}$ that outputs a polynomial $p(X_1, \dots, X_k)$ such that $p(x_1, \dots, x_k) = y$. Under the AGM, $\mathcal{A}$ outputs C together with coefficients c and $\{\tilde{c}_r^{(u)}\}_{u \in [s], r \in [k]}$, where

$$C = c \times G + \sum_{u \in [s], r \in [k]} \langle \tilde{c}_r^{(u)}, \mathbf{H}_r^{(u)} \rangle.$$

$\mathcal{E}$ observes $\{\tilde{c}_r^{(u)}\}_{u \in [s], r \in [k]}$ and outputs $p(X_1, \dots, X_k) := (\tilde{c}_1^{(s)}, \dots, \tilde{c}_k^{(s)}) \otimes (X_1, \dots, X_k)$. Then, we must show that $p(X_1, \dots, X_k)$ is consistent with the tuple $((x_1, \dots, x_k), y, \pi)$.

Let us parse the opening proof π as $(\{\mathbf{D}_j\}_{j \in [k-1]}, \mathbf{T}_k)$. Under the AGM, there exist coefficients $\{d_{j,i}\}_{j \in [k-1], i \in [d]}$ and $\{\tilde{d}_{r,j,i}^{(u)}\}_{u \in [s], r \in [k], j \in [k-1], i \in [d]}$ such that, for each $j \in [k-1]$ and $i \in [d]$,

$$D_j[i] = d_{j,i} \times G + \sum_{u \in [s], r \in [k]} \langle \tilde{d}_{r,j,i}^{(u)}, \mathbf{H}_r^{(u)} \rangle.$$

Since the verifier accepts π , it holds that $e(C, V) = \sum_{i \in [d]} e(D_1[i], V_{i,1}^{(s)})$. In other words,

$$e(c \times G + \sum_{u \in [s], r \in [k]} \langle \tilde{c}_r^{(u)}, \mathbf{H}_r^{(u)} \rangle, V) = \sum_{i \in [d]} e(d_{1,i} \times G + \sum_{u \in [s], r \in [k]} \langle \tilde{d}_{r,1,i}^{(u)}, \mathbf{H}_r^{(u)} \rangle, \mu_{i,1}^{(s)} \times V).$$

Observing that every term on the right-hand side has a factor of $\mu_{i,1}^{(s)}$, we conclude that $c = 0$, $\tilde{c}_r^{(u)} = 0$ for all $u \neq s$, and $\tilde{c}_r^{(s)} = 0$ for all $r \neq 1$; otherwise, $\mathcal{A}$ would break the DLog assumption. Then, we have $\tilde{d}_{r,1,i}^{(u)} = 0$ for all $u \neq s$ and $r \notin \{1, 2\}$ and $d_{1,i} = 0$ for all $i \in [d]$ due to the DLog assumption. The term $(\mu_{i,1}^{(s)})^2$ appears only on the right-hand side through $\langle \tilde{d}_{1,1,i}^{(s)}, \mathbf{H}_1^{(s)} \rangle$. Therefore, we must have $\tilde{d}_{1,1,i}^{(s)} = 0$; otherwise, $\mathcal{A}$ would break the CDH assumption. Finally, note that for all $i \in [d]$ and all $t \neq i$, the t th entry of $\tilde{d}_{2,1,i}^{(s)}$ must be zero; otherwise, the product term $\mu_{i,1}^{(s)} \cdot \mu_{t,2}^{(s)}$ would appear on the right-hand side, yet it cannot be generated on the left-hand side under the CDH assumption. With the above implications, the verification equation would be simplified to:

$$e(\langle \tilde{c}_1^{(s)}, \mathbf{H}_1^{(s)} \rangle, V) = \sum_{i \in [d]} e(\langle \tilde{d}_{2,1,i}^{(s)}, \mathbf{H}_2^{(s)} \rangle, \mu_{i,1}^{(s)} \times V).$$

Then, by the DLog assumption, we must have $\tilde{c}_1^{(s)} = \sum_{i \in [d]} \tilde{d}_{2,1,i}^{(s)}$. The verifier, similarly, checks that $e(C_{j-1}, V) = \sum_{i \in [d]} e(D_j[i], V_{i,j}^{(s)})$ for all $j = 2, 3, \dots, k-1$ such that $C_j = \langle \vec{x}_j, \mathbf{D}_j \rangle$. Thus,

$$e(\langle \vec{x}_{j-1}, \mathbf{D}_{j-1} \rangle, V) = \sum_{i \in [d]} e(d_{j,i} \times G + \sum_{u \in [s], r \in [k]} \langle \tilde{d}_{r,j,i}^{(u)}, \mathbf{H}_r^{(u)} \rangle, \mu_{i,j}^{(s)} \times V).$$

We conclude that $\mathbf{D}_{j-1} = \langle \tilde{d}_{j,j-1,*}^{(s)}, \mathbf{H}_j^{(s)} \rangle$, as above, due to the DLog or CDH assumptions. Then,

$$e(\langle \vec{x}_{j-1}, \langle \tilde{d}_{j,j-1,*}^{(s)}, \mathbf{H}_j^{(s)} \rangle \rangle, V) = \sum_{i \in [d]} e(\langle \tilde{d}_{j+1,j,i}^{(s)}, \mathbf{H}_{j+1}^{(s)} \rangle, \mu_{i,j}^{(s)} \times V).$$

Thus, $\langle \vec{x}_{j-1}, \langle \tilde{d}_{j,j-1,*}^{(s)}, \mathbf{H}_j^{(s)} \rangle \rangle = \langle \sum_{i \in [d]} \tilde{d}_{j+1,j,i}^{(s)}, \mathbf{H}_j^{(s)} \rangle$, and so, $\sum_{i \in [d]} \tilde{d}_{j+1,j,i}^{(s)} = \langle \vec{x}_{j-1}, \tilde{d}_{j,j-1,*}^{(s)} \rangle$. Finally, since $C_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k \rangle = \langle \vec{x}_{k-1}, \langle \tilde{d}_{k,k-1,*}^{(s)}, \mathbf{H}_k^{(s)} \rangle \rangle$, we obtain $\mathbf{T}_k = \langle \vec{x}_{k-1}, \tilde{d}_{k,k-1,*}^{(s)} \rangle$. Hence, $\mathbf{T}_k = (\tilde{c}_1^{(s)}, \dots, \tilde{c}_{k-1}^{(s)}) \otimes (x_1, \dots, x_{k-1})$ and $\langle \mathbf{T}_k, \vec{x}_k \rangle = y$. This completes the proof. $\square$

Below, we recall a lemma of Hafezi et al. [Haf+25] which shows that zk-KZH satisfies 2-special soundness if KZH is knowledge sound. Since the extractor for zk-KZH only rewinds the adversary and does not rely on the AGM, their result naturally extends to the updatable-SRS setting.

Lemma C.3. [Haf+25, Lemma 3] *zk-KZH satisfies 2-special soundness in the updatable-SRS setting under the knowledge soundness of KZH in the updatable-SRS setting.*

Lemma C.4. *zk-Spartan + zk-KZH satisfies special soundness in the updatable-SRS setting.*

Proof. We can extract a witness and construct a tree of accepting transcripts by invoking the tree-extractor for zk-KZH. Lemma C.2 guarantees that this procedure succeeds with overwhelming probability. Moreover, since Spartan is a sound PIOP [Set20, Lemma A.1], the extracted witness is satisfiable, and the resulting transcript tree is accepting. Soundness of the zero-knowledge Spartan PIOP (built from zero-knowledge sumchecks) was shown by Chen et al. [Che+23, Theorem A.3]. $\square$

Next, we show that zk-Spartan+zk-KZH satisfies the TLZK property.

Lemma C.5. *Spartan + zk-KZH with r rounds satisfies $(r-1)$ -trapdoor-less zero-knowledge.*

Proof. Both zk-Spartan (built from zero-knowledge sumchecks) and zk-KZH are zero-knowledge, and hence, their composition is also zero-knowledge [Chi+20]. The simulator for zk-Spartan+zk-KZH invokes the zk-KZH simulator to generate commitments and openings, and invokes the Spartan simulator to generate the sumcheck messages. The simulator for Spartan is information-theoretic and neither uses any trapdoor nor reprograms the random oracle. The simulator for KZH does not employ any trapdoor, but it reprograms the random oracle once when simulating the opening proof. Since openings are invoked in round $r - 1$, zk-Spartan+zk-KZH is $(r - 1)$ -TLZK. $\square$

Finally, we need show that zk-Spartan+zk-KZH satisfies the unique response property. For this purpose, we first prove that zk-KZH satisfies 1-unique response.

Lemma C.6. *zk-KZH satisfies 1-unique response.*

Proof. The opening procedure of zk-KZH consists of two rounds. In the first round, the prover sends a commitment to a masked polynomial and its evaluation at the query point, and the verifier issues a challenge. In the second round, the prover provides a KZH opening proof for the masked polynomial. Therefore, to show that zk-KZH satisfies 1-unique response, it suffices to show that, for a fixed commitment and evaluation point, KZH admits a unique opening proof.

Suppose there exist two distinct openings (y, π) and (y^*, π^*) for a commitment C at the point $(\vec{x}_1, \dots, \vec{x}_k)$. Since both openings are accepted, the following three conditions must hold:

- $\forall j \in [k - 1]$, we have $\sum_{i \in [d]} e(D_j[i], \mathbf{V}_{i,j}^{(s)}) = e(C_{j-1}, V) = \sum_{i \in [d]} e(D_j^*[i], \mathbf{V}_{i,j}^{(s)})$. By the Setup-Find-Repo assumption [DRZ20], it follows that $D_j[i] = D_j^*[i]$ for all $i \in [d]$ and $j \in [k - 1]$.
- $C_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k^{(s)} \rangle = \langle \mathbf{T}_k^*, \mathbf{H}_k^{(s)} \rangle$. By the DLog assumption, we must have $\mathbf{T}_k = \mathbf{T}_k^*$. Together with the previous item, this implies that $\pi = \pi^*$.
- $y = \langle \mathbf{T}_k, \vec{x}_k \rangle = \langle \mathbf{T}_k^*, \vec{x}_k \rangle = y^*$, contradicting the assumption that (y, π) and (y^*, π^*) are distinct.

$\square$

Then, we can use Lemma C.6 to show that zk-Spartan+zk-KZH satisfies $(r - 1)$ -unique response, where r denotes the total number of rounds.

Lemma C.7. *zk-Spartan+zk-KZH with r rounds satisfies $(r - 1)$ -unique response.*

Proof. In the r th round, zk-Spartan+zk-KZH opens a KZH commitment at a random evaluation point. By Lemma C.6, KZH satisfies 1-unique response, and thus the opening proof is unique. Therefore, zk-Spartan+zk-KZH satisfies $(r - 1)$ -unique response. $\square$

Lemmas C.1, C.4, C.5, and C.7 imply that zk-Spartan+zk-KZH is simulation extractable.

Theorem 2. *zk-Spartan+zk-KZH satisfies simulation extractability.*

C.2 Simulation Extractability of Spartan+KZH-Fold

We show that our PCD scheme based on zk-Spartan+zk-KZH is simulation extractable. To this end, we first formalize simulation extractability for leaf-decoupled PCD schemes. The definition of simulation extractability for arguments (see Definition C.1) does not directly extend to PCD schemes, since an adversary may interleave simulated proofs with honest proofs to produce an accepting transcript. In this setting, extraction from simulated proofs may be infeasible.

Abdolmaleki et al. [Abd+25] proposed a definition of simulation extractability for PCD schemes in which the extractor is only required to extract witnesses from non-simulated proofs generated by the adversary. However, their definition does not apply to our leaf-decoupled Spartan+KZH-Fold PCD scheme. In particular, their framework assumes that internal nodes are non-deterministic, and they output zero-knowledge accumulators (with respect to the input witness). The adversary is given oracle access to the PCD simulator, which can simulate proofs for both leaves and internal nodes. In our construction, internal nodes are deterministic: they only accumulate two input proofs and carry no additional witness. Therefore, internal nodes need not be zero-knowledge, and then, we may not have an efficient simulator algorithm that can forge internal-node proofs on the fly.

To address this issue, we relax the simulator in the definition of simulation extractability so that it only answers PCD leaf queries. Since leaves in our PCD scheme are zk-Spartan+zk-KZH proofs, they are zero-knowledge and can be simulated. Moreover, in the simulation-based security analysis of WillowFold, the simulator only needs to simulate client proofs, which are zk-Spartan+zk-KZH proofs. Internal-node proofs can then be generated by invoking the KZH-Fold accumulator.

Below, we recall the definition of simulation extractability for PCD schemes from Abdolmaleki et al. [Abd+25], adapted to the leaf-decoupled setting. To state the definition, we first introduce the notion of partially compliant PCD transcripts.

Definition C.6 (Partially Compliant PCD Transcripts [Abd+25]). *Let PCDT be a PCD transcript, let V be a PCD verifier algorithm, and let Q_{Sim} denote the set of queries made to the PCD simulator oracle Sim . We say that a vertex $u \in \text{PCDT.Nodes}$ is (V, Q_{Sim}) -compliant if, for every outgoing edge (u, v) labeled by (z, π) , the following conditions hold unless $(z, \pi) \in Q_{\text{Sim}}$.*

- *The verifier accepts the outgoing proof, i.e., $V(\text{pp}, z, \pi) = 1$.*
- *Let $e^{(1)}, \dots, e^{(n)}$ be the incoming edges to u , where each edge $(e^{(i)}, u)$ is labeled by $(z^{(i)}, \pi^{(i)})$. Let W denote the local witness at node u , and let R_u be the relation associated with u . Then it holds that $R_u(z, z^{(1)}, \dots, z^{(n)}, w) = 1$.*

A PCD transcript is (V, Q_{Sim}) -compliant if all of its vertices are (V, Q_{Sim}) -compliant.

Definition C.7 (Simulation Extractability of Leaf-Decoupled PCD). *Let PCD be a leaf-decoupled PCD scheme with a leaf simulator Sim^{leaf} . We say that PCD satisfies simulation extractability with respect to Sim^{leaf} if there exists an extractor $\mathcal{E}$ such that, for every efficient adversary $\mathcal{A}$,*

$$\text{Sim-Ext-PCD}_{0, \Pi}^{\text{Sim}^{\text{leaf}}, \mathcal{A}}(1^\lambda) \approx_c \text{Sim-Ext-PCD}_{1, \Pi, R}^{\mathcal{E}, \text{Sim}^{\text{leaf}}, \mathcal{A}}(1^\lambda).$$

The following lemma shows that a leaf-decoupled PCD built from a simulation extractable argument and a knowledge sound accumulation scheme satisfies simulation extractability.

Lemma C.8. *Let Π_{leaf} be a simulation extractable and Π_{internal} be a knowledge sound non-interactive argument. Let ACC_{leaf} and $\text{ACC}_{\text{internal}}$ be knowledge sound accumulation schemes for the verifier statements of Π_{leaf} and Π_{internal} , respectively. The Leaf-decoupled PCD scheme built from $(\Pi_{\text{leaf}}, \text{ACC}_{\text{leaf}}, \Pi_{\text{internal}}, \text{ACC}_{\text{internal}})$ —where Π_{leaf} is used at the leaves and Π_{internal} at internal nodes—satisfies leaf-SE.*

Proof. Our proof follows the approach of Abdolmaleki et al. [Abd+25], with the key difference that our simulator is weaker and only answers leaf queries. For each layer $i \in [d]$ of the PCD transcript, we construct an extractor $\mathcal{E}_i$ that extracts a PCD transcript of depth i . We show how to build $\mathcal{E}_{i+1}$ from $\mathcal{E}_i$ for each $i \in [d-2]$ using the extractor of the accumulation scheme. Finally, we combine $\mathcal{E}_{d-1}$ with the extractor of the NARK to obtain $\mathcal{E}_d$, which outputs witnesses for all non-simulated

Game Sim-Ext-PCD$_{0,\Pi}^{\text{Sim}^{\text{leaf}},\mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $(z, \pi) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H},\text{Sim}^{\text{leaf}}}(\text{pp})$ • $b \leftarrow V^{\mathcal{H}}(\text{pp}, z, \pi)$ • Return $b \wedge ((z, \pi) \notin Q_{\text{Sim}^{\text{leaf}}})$ <u>$V^{\mathcal{H}}(\text{pp}, z, \pi)$</u> • If $\text{isBase}(z) = 1$ return $V_{\text{leaf}}^{\mathcal{H}}(\text{pp}, z, \pi)$ • Else return $V_{\text{internal}}^{\mathcal{H}}(\text{pp}, z, \pi)$	Game Sim-Ext-PCD$_{1,\Pi,R}^{\mathcal{E},\text{Sim}^{\text{leaf}},\mathcal{A}}(1^\lambda)$ • $\text{pp} \leftarrow \Pi.\text{Setup}(1^\lambda, R)$ • $(z, \pi) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H},\text{Sim}^{\text{leaf}}}(\text{pp})$ • $b \leftarrow V^{\mathcal{H}}(\text{pp}, z, \pi)$ • $\text{PCDT} \leftarrow \mathcal{E}^{\mathcal{A}}(\text{pp}, z, \pi, Q_{\mathcal{O}}, Q_{\mathcal{H}}, Q_{\text{Sim}^{\text{leaf}}})$ • $b_{\text{out}} \leftarrow (z = \text{PCDT}.\text{Output})$ • $b_{\text{comp}} \leftarrow \text{IsComp}(\text{PCDT}, Q_{\text{Sim}^{\text{leaf}}})$ • Return $b \wedge b_{\text{out}} \wedge b_{\text{comp}} \wedge ((z, \pi) \notin Q_{\text{Sim}^{\text{leaf}}})$ <u>$V^{\mathcal{H}}(\text{pp}, z, \pi)$</u> • If $\text{isBase}(z) = 1$ return $V_{\text{leaf}}^{\mathcal{H}}(\text{pp}, z, \pi)$ • Else return $V_{\text{internal}}^{\mathcal{H}}(\text{pp}, z, \pi)$
--	---

Figure 16: The leaf-decoupled PCD simulation extractability games. Sim^{leaf} denotes the simulator of leaf nodes. IsComp is an efficient algorithm that checks whether PCDT is $(V^{\mathcal{H}}, Q_{\text{Sim}})$ -compliant.

leaf nodes of the PCD tree. The extractor $\mathcal{E}_0$ is simply the adversary, which outputs a tuple (z, π) . Let $\text{PCDT}_0 := (z, \pi)$. Then, for each $i \in [d-3]$, $\mathcal{E}_{i+1}$ proceeds as follows:

- Run $\mathcal{E}_i$ to obtain PCDT_i , a PCD transcript of depth i .
- For each node u at depth i of PCDT_i , run $\mathcal{E}^{\text{ACC}_{\text{internal}}}$, the extractor for $\text{ACC}_{\text{internal}}$, to extract all children of u . Since $\text{ACC}_{\text{internal}}$ is knowledge sound, this succeeds with overwhelming probability.
- Extend PCDT_i with the extracted children nodes to obtain PCDT_{i+1} . Output PCDT_{i+1} .

The extractor $\mathcal{E}_{d-1}$ proceeds as above; however, instead of invoking $\mathcal{E}^{\text{ACC}_{\text{internal}}}$, it invokes $\mathcal{E}^{\text{ACC}_{\text{leaf}}}$ to extract the leaf nodes and add them to the PCD transcript. Finally, $\mathcal{E}_d$ first runs $\mathcal{E}_{d-1}$ to obtain PCDT_{d-1} and then runs $\mathcal{E}^{\Pi_{\text{leaf}}}$ on each leaf node to extract a witness. Each extractor $\mathcal{E}_i$ invokes Sim^{leaf} to answer the adversary's simulator queries. Since Π_{leaf} is simulation extractable, the above extraction succeeds with overwhelming probability. $\square$

Our variant of Spartan+KZH-Fold constructed from the zk-Spartan+zk-KZH argument and the KZH-Fold accumulator. SE of zk-Spartan+zk-KZH was shown in Theorem 2. Knowledge soundness of KZH-Fold was shown in [Kad+25]. Since their extractor only rewinds the adversary and does not rely on the AGM, their proof extends to the updatable-SRS setting.

Lemma C.9. [Kad+25, Theorem 6] *The KZH-Fold accumulation scheme satisfies knowledge soundness in the updatable-SRS setting.*

Lemmas C.8 and C.9 and Theorem 2 imply that Spartan+KZH-Fold is simulation extractable.

Theorem 3. *Leaf-decoupled Spartan+KZH-Fold satisfies leaf simulation extractability.*

Game $\text{CCI}_{b_i(\text{AHE}, \Pi)}^{\mathcal{D}, \mathcal{A}}(1^\lambda)$

1. Run $\text{pp} \leftarrow \text{AHE.Setup}(1^\lambda)$.
2. Run $(\text{pk}, \{\text{sk}_j\}_{j \in [k]}) \leftarrow \text{AHE.KeyGen}(\text{pp})$.
3. Set $\forall i \in [n], \forall c \in \{0, 1\} : m_{i,c} \leftarrow \mathcal{K}, r_{i,c} \leftarrow \mathcal{R}$.
4. Set $\forall i \in [n] : (\text{ct}_i, \pi_i) \leftarrow \text{AHE.VerifiableEnc}(\text{pp}, \text{pk}, m_{i,b}; r_{i,b})$.
5. Run $\{(\text{ct}_i^*, \pi_i^*)\}_{i \in [\ell]} \leftarrow \mathcal{A}(\text{pp}, \{(\text{ct}_i, \pi_i)\}_{i \in [n]}, \text{pk})$.
6. **For** $i = 1, 2, \dots, \ell$:
 - (a) **If** $(\text{ct}_i^*, \pi_i^*) \in \{(\text{ct}_i, \pi_i)\}_{i \in [n]}$ **set** $m_i^* = \perp$.
 - (b) **Else set** $m_i^* \leftarrow \text{VDec}_{\text{pp}\Pi}(\text{pp}, \text{ct}_i^*, \pi_i^*, \text{pk}, \{\text{sk}_j\}_{j \in [k]})$
 - (c) Let $M \subseteq \{m_i^*\}_{i \in [\ell]}$ with all message $m_i^* \neq \perp$.
7. **Return** $\mathcal{D}(\{m_{i,0}, m_{i,1}\}_{i \in [n]}, M)$

$\text{VDec}_{\text{pp}\Pi}(\text{pp}, \text{ct}, \pi, \text{pk}, \{\text{sk}_j\}_{j \in [k]})$

1. **If** $\text{AHE.VerifyEnc}(\text{pp}, \text{pk}, \text{ct}, \pi) = 0$ **return** $\perp$
2. $\forall j \in [k] : \text{pd}_j \leftarrow \text{AHE.ParDec}(\text{pp}, \text{ct}, \text{sk}_j)$
3. $m \leftarrow \text{AHE.Recover}(\text{pp}, \text{ct}, \{\text{pd}_j\}_{j \in [k]})$
4. **Return** m^*

Figure 17: Ciphertext-correlation intractability game.

D Security Proofs of WillowFold

In this section, we first formalize ciphertext correlation intractability, which rules out correlated ciphertext attacks. Then, we present the hybrid argument for the WillowFold protocol.

D.1 Ciphertext Correlation Intractability

We formally define ciphertext correlation intractability, which ensures that an adversary who obtains a set of AHE ciphertexts, each accompanied by a proof of plaintext knowledge, cannot forge fresh satisfiable ciphertext-proof pairs whose plaintexts are correlated with the queried ciphertexts.

Definition D.1 (Ciphertext Correlation Intractability). *Let AHE be a threshold additively homomorphic encryption scheme with message space $\mathcal{M}$ and nonce space $\mathcal{R}$. Let Π be an argument for the relation $\mathcal{R}$*

$$\mathcal{R}_e = \{(\text{ct}, (m, r)) : \text{ct} = \text{AHE.Enc}(m, \text{pk}; r)\}$$

For any $n, \ell \in \mathbb{N}$, and a message distribution $\mathcal{K}$ over $\mathcal{M}$, we say that $\Phi := (\text{AHE}, \Pi)$ satisfies $(\mathcal{K}, n, \ell)$ -ciphertext correlation intractability (see Figure 17) if for every efficient adversary $\mathcal{A}$ and distinguisher $\mathcal{D}$,

$$\text{CCI}_{0, \Phi}^{\mathcal{D}, \mathcal{A}}(\lambda) \approx_c \text{CCI}_{1, \Phi}^{\mathcal{D}, \mathcal{A}}(\lambda).$$

Theorem 4. Assume AHE is an IND-CPA threshold additively homomorphic encryption scheme, and let Π be the underlying argument system used for proofs of plaintext knowledge such that Π satisfies simulation extractability. Then, for any $n, \ell \in \mathbb{N}$ and any distribution $\mathcal{K}$ over the plaintext domain, AHE satisfies $(\mathcal{K}, n, \ell)$ -ciphertext correlation intractability.

Proof. Let $(\mathcal{A}, \mathcal{D})$ denote efficient algorithms against CCI game with a non-negligible advantage ϵ_{CCI} . Then, there exists an efficient adversary $\mathcal{B}$ against the IND-CPA security of AHE. In particular, let $\mathcal{C}$ be the challenger in the IND-CPA security game of AHE, and Sim and $\mathcal{E}$ be the simulator and extractor in the Sim-Ext security of Π . Then, the reduction works as follows:

- Run $\text{pp}_\Pi \leftarrow \Pi.\text{Setup}(\lambda)$.
- Receive (pp, pk) from $\mathcal{C}$.
- $\forall i \in [n], \forall c \in \{0, 1\} : m_{i,c} \leftarrow \mathcal{K}$.
- Send $\{m_{i,c}\}_{i,c}$ to $\mathcal{C}$ and receive $\{\text{ct}_i\}_{i \in [n]}$.
- $\forall i \in [n] : \text{Run Sim on input } (\text{pp}, \text{ct}_i, \text{pk}) \text{ to simulate a proof } \pi_i. \text{ Record } (\text{pp}, \text{ct}_i, \text{pk}, \pi_i) \text{ in } Q_{\text{Sim}}.$
- Run $\{(\text{ct}_i^*, \pi_i^*)\}_{i \in [\ell]} \leftarrow \mathcal{A}(\text{pp}, \{(\text{ct}_i, \pi_i)\}_{i \in [n]}, \text{pk})$.
- For $i = 1, 2, \dots, \ell$:
 - If $(\text{ct}_i^*, \pi_i^*) \in \{(\text{ct}_i, \pi_i)\}_{i \in [n]}$ set $m_i^* = \perp$.
 - If π_i^* is not a valid proof for ct_i^* set $m_i^* = \perp$.
 - Else run $\mathcal{E}^{\mathcal{A}}$ on input (ct_i^*, π_i^*) to extract m_i^* .
 - Let $M \subseteq \{m_i^*\}_{i \in [\ell]}$ with all message $m_i^* \neq \perp$.
- Return $b^* \leftarrow \mathcal{D}(\{m_{i,0}, m_{i,1}\}_{i \in [n]}, M)$.

The simulated proofs π_i are indistinguishable from real ones, since Π is zero-knowledge. Extractor $\mathcal{E}^{\mathcal{A}}$ can extract m_i^* with overwhelming probability $\epsilon_{\text{Sim-Ext}}$. Thus, $\mathcal{B}$ has advantage $\epsilon_{\text{CCI}} - \epsilon_{\text{Sim-Ext}}$. $\square$

D.2 Hybrid Argument for WillowFold

To present the hybrid argument for WillowFold, we first recall the notion of key-sum leakage resistance—a property we require for the KAHE scheme.

Definition D.2 (Key-Sum Leakage Resistance). [BC+25] For any $n \in \mathbb{N}$, an KAHE scheme is n -semantic secure under leakage of key sum (Figure 18) if there exists an efficient simulator Sim s.t. for any inputs $(x_i)_{i \in [n]}$ and every efficient $\mathcal{A}$,

$$\text{Leakage}_{0, \text{KAHE}}^{\text{Sim}, \mathcal{A}}(\lambda, (x_i)_i) \approx_c \text{Leakage}_{1, \text{KAHE}}^{\text{Sim}, \mathcal{A}}(\lambda, (x_i)_i).$$

Theorem 5. Assume KAHE is a key- and message-homomorphic encryption scheme with key-sum leakage resistance, AHE is a threshold additively homomorphic encryption scheme, and Π_{PCD} is a leaf-decoupled PCD scheme built from a simulation extractable argument and a knowledge sound accumulation scheme. Then, the secure aggregation protocol of Figures 4, 5, and 6 realizes $\mathcal{F}_{\text{Agg}}$ in Figure 1 in the $(\text{RO}, \text{Setup})$ -hybrid model against a static active adversary corrupting the server, any number of clients, and $\leq 2t - k - 1$ decryptors, with AHE threshold t and committee size k .

Proof. The simulator is described in Section 6.2. We prove security of the scheme in the $(\text{RO}, \text{Setup})$ -hybrid model and assume AHE.KeyGen is realized via a secure distributed key generation (see [BC+25] for a formal proof of our DKG instantiation). Below, we only focus on the aggregation and give hybrids.

Game $\text{Leakage}_{0,\text{KAHE}}^{\text{Sim},\mathcal{A}}(\lambda, (x_i)_i)$ • $\text{pp} \leftarrow \text{KAHE.Setup}(1^\lambda)$ • $\forall i \in [n] : \mathbf{k}_i \leftarrow \text{KAHE.KeyGen}(\text{pp})$ • $\forall i \in [n] : \mathbf{c}_i \leftarrow \text{KAHE.Enc}(\text{pp}, x_i, \mathbf{k}_i)$ • $b \leftarrow \mathcal{A}(\text{pp}, \sum_i \mathbf{k}_i, (\mathbf{c}_i)_i)$ • Return b	Game $\text{Leakage}_{1,\text{KAHE}}^{\text{Sim},\mathcal{A}}(\lambda, (x_i)_i)$ • $\text{pp} \leftarrow \text{KAHE.Setup}(1^\lambda)$ • $(\mathbf{k}, (\mathbf{c}_i)_i) \leftarrow \text{Sim}(\text{pp}, \sum_i x_i)$ • $b \leftarrow \mathcal{A}(\text{pp}, \mathbf{k}, (\mathbf{c}_i)_i)$ • Return b
---	--

Figure 18: The key sum leakage resistance game.

Proof Overview. There are a couple of differences between the real world and the ideal world. First, the random oracle is reprogrammed. However, since seeds are encrypted and only revealed after reprogramming, this remains indistinguishable from the real world. Second, Sim simulates honest clients with input 0^ℓ . This is also indistinguishable from the real world because: (i) KAHE is secure even under leakage of the sum of the keys; (ii) AHE is ciphertext correlation intractable; therefore the adversary's keys are uncorrelated with honest clients' keys and do not change their sum distribution which implies that it is calibrated correctly to invoke the leakage-resilient property of the KAHE scheme; and (iii) PCD leaves are zero-knowledge and keep clients' inputs hidden.

Note that zero-knowledge is required only for PCD leaves, since internal nodes are deterministic and carry no witness. The same holds for simulation extractability as discussed in 6.2.

The hybrids are similar for malicious and semi-honest settings; they differ only in how Sim recovers the set of included honest clients S_H , to subsequently retrieve $\sum_{i \in S_H} \vec{x}_i$ from the ideal functionality, and reprogram the random oracle, which has been described in Section 6.2. For simplicity, in the sequel we assume that the server does not drop or replace any honest contribution, i.e., $H = S_H$ as the simulator can extract S_H from PCD leaves.

Hybrid Argument. $\mathcal{A}$ may corrupt the server, any number of clients, and $\leq 2t - k - 1$ decryptors.

- $\text{Hyb}^{(0)}$: The real execution, and $\mathcal{A}$'s view include honest contributions

$$(\text{ct}_i^{\text{key}}, \text{ct}_i^{\text{seed}}, \text{ct}_i^{\text{masked}}, \Pi_i^{\text{PCD}})$$

and honest partial decryptions pd_j . It outputs (rt, g_i, π_i) to honest clients and

$$(\text{encKey}, g_{\text{pv}}, \Pi^{\text{PCD}}), \Pi^{\text{ID}}$$

to honest decryptors.

- $\text{Hyb}^{(1)}$: The same as $\text{Hyb}^{(0)}$, except that we set one of the honest client's input to $\sum_{i \in H} \vec{x}_i$ and other honest clients' inputs to 0^ℓ . $\text{Hyb}^{(1)}$ is indistinguishable from $\text{Hyb}^{(0)}$ since we know that KAHE is n -semantically secure under key sum leakage and hides the clients' inputs. Note that AHE is ciphertext correlation intractable due to Theorem 4, which implies that the keys $\{k_i\}_{i \in C}$ used by corrupted parties are independent of the keys $\{k_i\}_{i \in H}$ used by honest clients. Hence, the adversary learns nothing about $\{k_i\}_{i \in H}$ beyond $\sum_{i \in H} k_i$, which is allowed by KAHE.

- $\text{Hyb}^{(2)}$: The same as $\text{Hyb}^{(1)}$, except that we set all honest clients' input to 0^ℓ and reprogram the RO so that $\mathcal{H}(\text{seed}_h) \mapsto \sum_i \vec{x}_i$ for one of the honest clients $h \in H$. Since seeds are sampled uniformly at random and AHE is semantically secure, the probability that the adversary ever invokes PRG on input seed_h before reprogramming is negligible. Thus, $\text{Hyb}^{(2)}$ is indistinguishable from $\text{Hyb}^{(1)}$. See Section 6.2 for further details on reprogramming.
- $\text{Hyb}^{(3)}$: The same as $\text{Hyb}^{(2)}$, except that the partial decryptions are replaced with the output of the AHE simulator. By the security of threshold decryption in the AHE scheme, $\text{Hyb}^{(3)}$ is indistinguishable from $\text{Hyb}^{(1)}$.

$\text{Hyb}^{(3)}$ almost matches the ideal world simulated by Sim up to a slight left difference: Sim aborts if any duplicate IDs appear in the leaf proofs. By simulation extractability of Π^{PCD} and soundness of Π^{ID} , the real-world execution would also abort in this case. Therefore, $\text{Hyb}^{(3)}$ is indistinguishable from the ideal world. $\square$

E Concrete Instantiations of Encryption Schemes

For both the AHE and KAHE we use the same instantiation of Willow based on the RLWE assumption. We denote $R_q = \mathbb{Z}[X]/(q, X^N + 1)$, the RLWE ring consisting of all polynomials with integer coefficients reduced modulo q , with addition and multiplication performed modulo both q and the polynomial $X^N + 1$. Equivalently, each element of R_q is uniquely represented by a polynomial of degree less than N with coefficients in $\mathbb{Z}_q$. Concrete choices of q and N for which the RLWE assumption is believed to hold are obtained using lattice-estimation tools [APS15]. In our work, we adopt the same parameter sets as those used in Willow (Table 2 in Willow), since our experiments are conducted under identical parameter choices. Here $\|\cdot\|$ is the infinity norm.

RLWE-Based KAHE Scheme. Let $N_1 > 0$ be a power of two, and let $R_{q_1} = \mathbb{Z}[X]/(q_1, X^{N_1} + 1)$ for integer $q_1 > 0$. Let $t_1 > 0$ be an integer coprime to q_1 ; the plaintext space in our KAHE scheme is $R_{t_1} = \mathbb{Z}[X]/(t_1, X^{N_1} + 1) \equiv \mathbb{Z}_{t_1}^{N_1}$. Let $\sigma_s, \sigma_e > 0$ denote the Gaussian parameters for the secret and error distributions. Specifically, D_{σ_s} and D_{σ_e} output elements in R_{q_1} , where each entry is independently sampled from a Gaussian distribution with mean 0 and standard deviation σ_s and σ_e , respectively. Then:

- KAHE.Setup: Samples and returns $a \leftarrow R_{q_1}$ as the public parameter, which is implicit to the following algorithms.
- KAHE.KeyGen: Samples and returns $k \leftarrow D_{\sigma_s}$.
- KAHE.Enc: Samples $e \leftarrow D_{\sigma_e}$, and returns $c = a \cdot k + t \cdot e + x \in R_{q_1}$.
- KAHE.Dec: Returns $(c - a \cdot k) \bmod t_1$.

RLWE-Based AHE Scheme. We instantiate AHE under the standard RLWE assumption, and augment it with ZKPoK systems Π_{KeyGen} and Π_{Enc} . Let $N_2 > 0$ be a power of two, and let $R_{q_2} = \mathbb{Z}[X]/(q_2, X^{N_2} + 1)$ for an integer modulus $q_2 > 0$. Let $t_2 > 0$ be an integer such that the plaintext space is $\mathbb{Z}[X]/(t_2, X^{N_2} + 1) \equiv \mathbb{Z}_{t_2}^{N_2}$, and let $\Delta = \lfloor q_2/t_2 \rfloor$ be a scaling factor. Let $\chi_s, \chi_e, \chi_{\text{flood}}$ be distributions over R_{q_2} . Our AHE consists of the following algorithms:

- AHE.Setup(1^λ) = $(\mathbf{u}, \text{zk}_{\text{params}})$: Samples $\mathbf{u} \leftarrow R_{q_2}$, and generate the public parameters $\text{zk}_{\text{params}}$ of the ZKPoK systems; they are implicit in the following algorithms.

- $\text{AHE.VerifiableKeyGen}((r_1, r_2)) = (\text{sk}_j, \text{pk}_j, \pi_{\text{KeyGen}})$: Samples $\text{sk}_j \leftarrow \chi_s(r_1)$ and $e_j \leftarrow \chi_e(r_2)$, sets $\text{pk}_j = -\mathbf{u} \cdot \text{sk}_j + e_j$, and generates a ZK proof $\pi_{\text{KeyGen}} \leftarrow \text{Prove}(\text{pk}_j, (\text{sk}_j, e_j))$ for the relation

$$R_{\text{KeyGen}} = \{(\text{pk}_j, \text{sk}_j, e_j) \mid \text{pk}_j = -\mathbf{u} \cdot \text{sk}_j + e_j \bmod q_2, \|\text{sk}_j\| \leq B_s, \|e_j\| \leq B_e\},$$

where B_s and B_e are parameters determined based on χ_s, χ_e and the proof system.

- $\text{AHE.VerifyKeyGen}(\pi_{\text{KeyGen}}, \text{pk}_j)$: Returns true if π_{KeyGen} verifies R_{KeyGen} over pk_j .
- $\text{AHE.KeyAgg}(\{\text{pk}_j\}_{j=1}^m)$: Returns the public key $\text{pk} = \sum_{j=1}^m \text{pk}_j \in R_{q_2}$.
- $\text{AHE.VerifiableEnc}(x, \text{pk}; r_1, r_2, \tau) = (\text{ct}^0, \text{ct}^1, \pi_{\text{Enc}})$: Samples $v \leftarrow \chi_s(r_1)$ and $e^0, e^1 \leftarrow \chi_e(r_2)$, and computes

$$\text{ct}^0 = \text{pk} \cdot \mathbf{v} + e^0 + \Delta \cdot x \in R_{q_2}, \quad \text{ct}^1 = \mathbf{u} \cdot \mathbf{v} + e^1 \in R_{q_2}.$$

Also generates a ZK proof π_{Enc} for the relation

$$R_{\text{Enc}} = \{(\text{ct}^1, \mathbf{v}, e^1) \mid \text{ct}^1 = \mathbf{u} \cdot \mathbf{v} + e^1 \bmod q_2, \|\mathbf{v}\| \leq B_s, \|e^1\| \leq B_e\},$$

- $\text{AHE.VerifyEnc}(\pi_{\text{Enc}}, \text{ct})$: Returns true if π_{Enc} verifies relation R_{Enc} over ct^1 .
- $\text{AHE.ParDec}(\text{ct}^1, \text{sk}_j) = \text{ct}^1 \cdot \text{sk}_j + e_{\text{flood}}$: Samples $e_{\text{flood}} \leftarrow \chi_{\text{flood}}$, and returns $\text{pd}_j = \text{ct}^1 \cdot \text{sk}_j + e_{\text{flood}} \in R_{q_2}$.
- $\text{AHE.Recover}((\text{ct}^0, \text{ct}^1), \{\text{pd}_j\}_{j=1}^m) = x$: Returns

$$x = \left\lfloor \frac{\text{ct}^0 + \sum_{j=1}^m \text{pd}_j}{\Delta} \right\rfloor.$$

Encryption Scheme E. Our protocol uses an additional asymmetric encryption scheme **E** that is required to satisfy IND-CPA security. This encryption scheme is not required to satisfy any further properties. Similar to Willow, we instantiate it using the same AHE scheme.