WhiteCloak: How to Hold Anonymous Malicious Clients Accountable in Secure Aggregation?

Zhi Lu[a], Yongquan Cui[a], Songfeng Lu[*ab]
[a]School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China
[b]Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen, China

***Abstract*—With the advancement of artificial intelligence and the increasing digitalization of various sectors, the scale of personal data collection and analysis continues to grow, leading to heightened demands for privacy protection of personal data and identity. However, existing secure aggregation methods, such as ACORN (USENIX 2023), while ensuring the privacy and compliance of input data, fail to meet the requirements for client anonymity. Simply applying anonymous credentials allows previously identified malicious clients (e.g., those using non-compliant data) to re-enter aggregation rounds by updating their credentials, thus evading accountability. To address this issue, we propose WhiteCloak, the first secure aggregation solution that ensures accountability under client anonymity. WhiteCloak requires each client i to participate in round τ using an anonymous credential $\widetilde{i}_\tau$. Before participation, each client must submit a zero-knowledge proof verifying that they have not been blacklisted, preventing malicious clients from evading accountability by changing their credentials. WhiteCloak can be seamlessly integrated into existing frameworks. In federated learning experiments on the SHAKESPEARE dataset, WhiteCloak adds only 1.77s of additional processing time and 35.68KB of communication overhead, accounting for 0.34% and 0.1% of ACORN's total overhead, respectively.**

I. Introduction

In many practical applications, data aggregation is a core task that supports data-driven decision-making [1], such as in statistical analyses within healthcare, education, voting, and advertising, where machine learning enhances the client experience. However, most clients are unwilling to disclose their privacy, which is also protected by law [2].

Secure Aggregation (SA), such as those adopted by companies like Apple [1], Facebook [3], and Google [4], are designed to protect client input privacy while mitigating the risks of privacy leakage (e.g., inference attacks [5], [6], [7]), all while securely aggregating data from large-scale clients [8]. This technology has been widely implemented in areas such as recommendation engines [9] and time-series analysis [10]. Existing research, such as ACORN and RoFL [11], [12], [13], [14], [15], [16], [17], [18], extends SA by incorporating input validation, forming Secure Aggregation with Input Validation (SAIV), using technologies like homomorphic encryption, secure multi-party computation, and zero-knowledge proofs (ZKP) to verify data compliance and remove malicious clients (e.g., with data poisoning attacks [19], [20], [21]).

*Corresponding Author, {luzhi, yqcui1977, lusongfeng}@hust.edu.cn

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230142
www.ndss-symposium.org

Nevertheless, as the internet becomes more deeply integrated into everyday life, an increasing number of clients are not only unwilling to share plaintext data but also wish to maintain their anonymity. Research indicates that approximately 86% of clients prefer to access the internet anonymously [22], [23], [24]. As a result, anonymous credentials have emerged as an effective tool for safeguarding client identity (ID) privacy [25], and some data aggregation frameworks have started incorporating anonymous credentials to ensure client identity protection [26], [27].

However, a fundamental challenge arises when all clients utilize anonymous credentials for data aggregation: it becomes impossible to trace clients submitting illegal data, severely undermining legitimate accountability and regulatory measures [28]. This challenge originates from the inherent untraceability required by anonymity, wherein each client must update their identity in every aggregation round, preventing credentials from being linked across rounds.

Existing SAIV methods (e.g., ACORN [17], RoFL [16]) assume client identities are known, thus allowing servers to easily identify and permanently exclude malicious clients. When introducing anonymity, this identification mechanism fails completely—clients obtain new anonymous credentials every round, making banning impossible. To the best of our knowledge, no existing SAIV framework addresses the challenge of holding malicious clients accountable while preserving client anonymity across-round. This limitation poses significant risks, especially in sensitive real-world scenarios like federated healthcare or financial analysis, where anonymity and accountability must coexist.

To overcome this fundamental conflict between anonymity and cross-round accountability, we introduce a novel extension of the state-of-the-art SAIV framework ACORN [17], adapting it specifically to enable anonymous yet accountable client participation. We refer to this mechanism as the "WhiteCloak" which ensures that once a client's anonymous credential is blacklisted, it is excluded from future aggregations—even if it changes its credential each round. WhiteCloak achieves this by integrating three designed Groth16 proofs [29] alongside HICIAP [30]. These proofs verify (1) the legitimacy of the registered client ID, (2) the binding between the current credential and the registered identity, and (3) the client's non-blacklisted status. For any client flagged by ACORN as non-compliant, WhiteCloak blacklists their credential and removes both the corresponding input and secure aggregation key,

preventing further participation in subsequent rounds.

A. Main Contributions

The cost and functions brought by WhiteCloak are shown in table I. The contributions are as follows:

- **First SAIV system supporting cross-round accountability under anonymity:** Unlike existing schemes that assume known client identities, WhiteCloak ensures that even if an anonymous malicious client updates or fabricates credentials across rounds, it cannot evade blacklisting—yet its identity remains undisclosed.
- **Removal under anonymity:** In anonymous SAIV settings, a client's input and seed of key may be submitted under different anonymous credentials, making them unlinkable. WhiteCloak guarantees that once a credential is blacklisted, both the associated input and key are removed without recovering the real identity.
- **Misjudgment tolerance:** To prevent false positives, WhiteCloak adopts a multi-level blacklist that enforces banning only after exceeding a configurable threshold q. It also supports time-based automatic unblocking and clients can appeal their ban by proving their innocence.
- **Identification of incorrect-key clients:** Current SAIV systems (e.g., ACORN) can detect inconsistent keys but cannot localize the offending client. WhiteCloak introduces a new zero-knowledge proof construction that enables precise identification of anymous clients using incorrect keys.

To show that multiple SA application scenarios can benefit from the cross-round anonymous accountability introduced by WhiteCloak, we take federated learning as a representative scenario (Table I). With 500 clients and 25 adversaries on the q-th blacklist, WhiteCloak adds only 1.42 s of latency and 0.03 KB of per client (8.01 s and 5.70 KB in total). Because this overhead is independent of the dimensionality of the aggregated vectors, it remains virtually unchanged—never exceeding 1.8 s and 0.03 KB per client—on the higher-dimensional CIFAR-10L and Shakespeare tasks (Table II). Consequently, production systems that already rely on SA—such as mobile-telemetry platforms collecting anonymous statistics every few minutes, smart-meter infrastructures reporting hourly household energy consumption, and privacy-preserving census surveys aggregating demographic data across regions—can incorporate WhiteCloak with negligible additional bandwidth and latency while gaining the ability to block repeat offenders across rounds without revealing identities. When the blacklist contains at most 25 clients, the incremental cost consistently stays below 2 s and 0.1 KB per device, comfortably inside the performance budgets of current deployments.

B. Related Work

Secure Aggregation. Bonawitz et al. [32] proposed a dual-mask secure aggregation scheme that addresses the issue of clients going offline and reconnecting. Building on [32], Bell et al. [31] optimized the key generation scheme. Yiping et al. [33] introduced Flamingo, a method that supports reusing the same random seed across multiple rounds. Takeshita et al. [8] customized lattice encryption for secure aggregation. Karthikeyan et al. [34] reduce the number of communication rounds per aggregation. These studies focus on improving efficiency and reducing communication rounds, but do not address client-side input validation or anonymous client accountability.

Secure Aggregation with Input Validation. Chowdhury et al. [35] introduced EIFFeL, which uses SNIP [36] to verify client data, but with high overhead. Lycklama's RoFL [16], based on Bell et al.'s SA [31], uses Bulletproofs to validate client input norms, significantly reducing proof size and time. Yizheng et al. [18] introduced RiseFL, which reduces ZKP overhead through probabilistic checks. Bell et al. [17] further optimized this method by combining techniques from Gentry et al. [37] and Bulletproofs, greatly reducing proof size. Zhi et al. [38] proposed LZKSA, which utilized a specially constructed ZKP to accelerate the proof generation process. Some research has focused on multi-server settings. Henry et al. [36] and Surya et al. [12] proposed more efficient versions, Prio and Prio+, respectively. Mayank's ELSA [11] achieves both efficiency and malicious privacy protection by leveraging multi-party secure computation. Other methods [13], [14], [15] focus on weaker security models. Existing SAIV schemes lose cross-round accountability if real identities are replaced by anonymous credentials, because each round's credential is new and cannot be linked back to the same client.

Anonymous credential. Accountable anonymous credential schemes enable clients to anonymously prove possession of valid credentials while holding them accountable for misconduct through de-anonymization or blacklisting mechanisms. Joakim introduced PAPR [39], which implements publicly auditable privacy revocation, allowing clients to be de-anonymized upon audit. Markulf proposed Blueprint [28], which reveals the content of clients by comparing blueprint data on the watchlist with their actual data. Tsang et al. [40] introduced BLAC, which revokes credentials of malicious clients without de-anonymizing them. Michael developed SNARKBlock [30], employing a zero-knowledge blocklisting mechanism where clients submit proof of non-blacklisted membership, significantly reducing overhead.

Different from previous works, WhiteCloak adds anonymity to the SAIV system for the first time and solves the problem of "cross-round anonymous accountability" to be generated. It is the first solution to maintain client anonymity, pinpoint misbehavior, enforce persistent injunctions, and support grievance/recovery operations.

II. Preliminaries

A. Groth16

Groth16 is a zero-knowledge proof (ZKP) system that relies on a trusted setup [29]. It enables a prover to show that an arithmetic circuit is satisfied using both public and private inputs, and operates over a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where all groups share a prime order $p > 2^\lambda$, with security parameter λ. Due to its efficiency and compact proofs, Groth16 is well-suited for our fixed-circuit setting. Although alterna-

TABLE I: Comparison of WhiteCloak with State-of-the-Art SAIV Schemes. Parameters: data dimension ℓ, number of clients N, number of aggregated proofs n, elliptic curve pairing operation P, group element size $|\mathbb{G}|$, scalar field element size $\mathbb{F}$, variable base multi-scalar multiplication in the group M, and length of the q-th blacklist $|\mathsf{blk}^q|$.

Method	**Sum**	**SA** [31] (CCS 2020)	**RoFL** [16] (S&P 2023)	**ACORN-robust** [17] (USENIX 2023)	**WhiteCloak**
Communication	$\ell\|\mathbb{F}\|$	$(\ell+\log N)\|\mathbb{F}\|$	$(\ell+N)\|\mathbb{F}\|$	$(\ell+\log^2 N)\|\mathbb{F}\|$	$(\ell+\log^2 N)\|\mathbb{F}\|+\|\mathbb{G}_1\|$ $+\|\mathbb{G}_2\|+\log^{\|\mathsf{blk}^q\|}\|\mathbb{G}_T\|$
Computation	0	$\ell\log N$	ℓN	$\ell\log N+\log^2 N$	$\ell\log N+\log^2 N+M_{\mathbb{G}_T}$ $n(M_{\mathbb{G}_1}+M_{\mathbb{G}_2}+P)$
One round, N=500,ℓ=19k, $\|\mathsf{blk}^q\|=25$ *verify* ℓ_∞/ℓ_2*:*		1.4MB/4.5MB 40s/47s	5.67KB/5.68KB 6.59s/6.65s	5.70KB/5.71KB 8.01s/8.07s	
Privacy					
Ciphertext	✗	✓	✓	✓	✓
Anonymous	✗	✗	✗	✗	✓
Accountability	(Message and Key are private, client identity is anonymous)				
Identify Incorrect Message	✗	✗	✓	✓	✓
Detect Incorrect Key	✗	✗	✓	✓	✓
Identify Incorrect Key	✗	✗	✗	✗	✓
Forbid Malicious with Tolerance	✗	✗	✗	✗	✓
Cross-round Accountability	✗	✗	✗	✗	✓
Credential Update Resistance	✗	✗	✗	✗	✓

tives like Plonk and Halo2 support more flexible or setup-free constructions, they incur greater proof size and overhead.

- $\mathsf{Groth16.Setup}$ (Relation) $\rightarrow$ crs: Generates a common reference string (crs) for a given arithmetic circuit.
- $\mathsf{Groth16.Prove}(\text{crs}; \{a_i\}_{i=0}^{\ell}; \{a_i\}_{i=\ell+1}^{m}) \rightarrow \pi$: Produces a proof $\pi = ([\eta]_1, [\theta]_2, [\iota]_1)$ that the circuit is satisfied, where $\{a_0, \ldots, a_\ell\} \in \mathbb{F}$ are public inputs, and $\{a_{\ell+1}, \ldots, a_m\} \in \mathbb{F}$ are private inputs, where

$$\eta = \alpha + \sum_{i=0}^{m} a_i u_i(X) + r\delta \quad \theta = \beta + \sum_{i=0}^{m} a_i v_i(X) + s\delta$$

$$l = \sum_{i=\ell+1}^{m} \frac{a_i\left(\beta u_i(X) + \alpha v_i(X) + w_i(X)\right) + h(X)t(X)}{\delta} + \eta s + \theta r - rs\delta$$

- $\mathsf{Groth16.Prepare}(\text{crs}; \{a_{ij}\}_{j=1}^{t}) \rightarrow \hat{S}$: Aggregates any subset of public inputs into a single group element $\hat{S} = \sum_{j=1}^{t} a_{ij} W_{ij}$, where W_i are CRS values corresponding to the i-th wire of the circuit.
- $\mathsf{Groth16.Vfy}(\text{crs}; \pi; \{a_0\}_{i=0}^{\ell}) \rightarrow \{0,1\}$: Verifies the proof $\pi = (A, B, C)$ by checking the pairing relation:

$$e(A,B) = e([\alpha]_1, [\beta]_2) \cdot e(C, [\delta]_2) \cdot \prod_{i=0}^{\ell} e(a_i W_i, [\gamma]_2),$$

where $[\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_2$ are elements from the CRS.

B. Hidden Common Input Aggregate Proof (HICIAP)

HICIAP [30] is a ZKP that aggregates n Groth16 [29] proofs into a single proof, ensuring all proofs verify and share a concealed common input. See the Appendix for the detailed HICIAP protocol. HICIAP consists:

- $\mathsf{HICIAP.GenCk}(n) \rightarrow (\text{ck}; \text{srs})$: Generates a commitment key $(\text{ck}_1, \text{ck}_2, \text{ck}_3) \in \mathbb{G}_2^n \times \mathbb{G}_1^n \times \mathbb{G}_2$ and a structured reference string (srs) to aggregate Groth16 proofs.
- $\mathsf{HICIAP.Prove}((\text{ck}, \text{crs}); \hat{\mathbf{S}}; (a_0, \{\pi_i\}_{i=1}^{n-2})) \rightarrow (\hat{\pi}, o)$: Produces a proof that all Groth16 proofs π_i verify with respect to the common witness $a_0 \in \mathbb{F}$, input $\hat{S}_i \in \mathbb{G}_1$, and Groth16 CRS, along with an opening o to a_0.
- $\mathsf{HICIAP.Vfy}(\text{srs}; \text{com}_{\text{in}}) \rightarrow \{0,1\}$: Verifies the aggregate proof against the committed public input, or directly with prepared Groth16 inputs.
- $\mathsf{HICIAP.LinkProve}(\{\hat{\pi}_i\}_{i=1}^{t}; (a_0, \{o_i\}_{i=1}^{t})) \rightarrow \pi_{\text{link}}$: Produces a link proof showing the aggregated proofs share the witness a_0 using openings o_i.
- $\mathsf{HICIAP.LinkVfy}(\pi_{\text{link}}; \{\hat{\pi}_i\}_{i=1}^{t}) \rightarrow \{0,1\}$: Verifies the link proof for the aggregate proofs.

C. Secure Aggregation with Input Validation (SAIV)

SAIV [16], [17], [18], [35] is a technique that combines Secure Aggregation (SA) [8], [31], [32], [33], [34], [41], [42] and ZKP [29], [43], [44], [45] to ensure privacy-preserving data aggregation in distributed data aggregation scenarios [46], [47] while preventing malicious clients from submitting incorrect message. The SAIV protocol (based on ACORN [17]) consists of the following steps:

- $\mathsf{SAIV.Setup}() \rightarrow \{\mathbf{s}_i, \mathbf{r}_i, \{s_{ij}\}_j\}$: Each client i generates seed_i and seeds $\{\text{seed}_{ij}\}_j$ with clients j and shareing them as $[\text{seed}_{ij}]$ by Shamir Secret Share [48]. then computes the keys $\mathbf{r}_i = \mathsf{Prf}(\text{seed}_i), \mathbf{s}_{ij} = \mathbf{Prf}(\text{seed}_{ij})$. The encryption key $\mathbf{s}_i$ is computed as Eq. 1 with feature as Eq. 2:

$$\mathbf{s}_i = \mathbf{r}_i + \sum \delta_{ij}\mathbf{s}_{ij}, \delta_{ij} = \begin{cases} 1, & \text{if } i > j \\ -1, & \text{if } i < j \end{cases}. \tag{1}$$

$$\sum_i \mathbf{s}_i = \sum_i (\mathbf{r}_i + \sum_j \delta_{ij}\mathbf{s}_{ij}) = \sum \mathbf{r}_i. \tag{2}$$

- $\mathsf{SAIV.Enc}(\mathbf{m}_i, \mathbf{s}_i) \rightarrow \mathbf{y}_i$: Clients encrypt message $\mathbf{m}_i$ to get input $\mathbf{y}_i$ by Pedersen commitment [49] with generators $\mathbf{g}, \mathbf{h}$:

$$\mathbf{y}_i = \mathsf{Com}(\mathbf{m}_i, \mathbf{s}_i) = \mathbf{g}^{\mathbf{m}_i}\mathbf{h}^{\mathbf{s}_i}. \tag{3}$$

- $\mathsf{SAIV.Prove}(\mathbf{m}_i, \mathbf{s}_i, V) \to \pi_i$: Leveraging Bulletproofs [43] based on Gentry et al. [37], generates proof π_i for relations R_{key}, R_{norm}, verifying constraints such as $\|\mathbf{m}_i\|_\infty \leq V$. Each proof π_i comprises commitments (A, S, T_1, T_2), challenges (b, z, x), and responses $(\tau_x, \mu, \hat{t}, l, r)$.
- $\mathsf{SAIV.Vfy}(\mathbf{y}_i, \pi_i) \to \{0, 1\}$: Computes $h'_i = h_i^{(b^{-i+1})}$, $P = AS^x \mathbf{g}^{-z} \mathbf{h'}^{z\mathbf{b}^n + z^2 \mathbf{2}^n}$. Here $\mathbf{2^n} = \{2^0, \cdots, 2^{n-1}\}$ and verifies: $\hat{t} \stackrel{?}{=} \langle l, r \rangle$, $P \stackrel{?}{=} \mathbf{h}^\mu \mathbf{g}^l \mathbf{h'}^r$, $\mathbf{g}^{\hat{t}} \mathbf{h}^{\tau_x} \stackrel{?}{=} V^{z^2} g^{\sigma(b,z)} T_1^x T_2^{x^2}$. If all checks pass, outputs 1; otherwise, outputs 0.
- $\mathsf{SAIV.Agg}(\{\mathbf{y}_i, \mathbf{r}_i, [\text{seed}_{di}]_{d \in \mathcal{U}_{\text{drop}}}\}_i) \to \mathbf{a}$: Server reconstruct the dropped client's key $\mathbf{s}_{dj}$ by $[\text{seed}_{dj}]$. Then completes the aggregation and decrypts the combined result:

$$\mathbf{a} = \sum_{i \in \mathcal{U}_{\text{online}}} (\mathbf{y}_i - \mathbf{r}_i) - \sum_{d \in \mathcal{U}_{\text{drop}}} \left(\sum_j \delta_{dj} \mathbf{s}_{dj} \right). \quad (4)$$

D. Threat model and Security requirements

WhiteCloak adopts the standard SAIV threat model shared by ACORN-robust and related systems [16], [17], [18], [35]: a single semi-honest server and n malicious clients. Building on this baseline, we incorporate the accountability-preserving anonymity guarantees of credential-based schemes [30], [40]. As [16] demonstrates, input validation neutralizes both single- and multi-round attacks, so WhiteCloak matches ACORN's model utility while uniquely enabling the exclusion of anonymous repeat offenders—something ACORN and its predecessors cannot. The formal threat model is defined as follows:

Definition 1 (Semi-honest server)**.** *The server follows the protocol exactly, accepts only well-formed inputs/credentials and rejects any that violate the public constraints. Yet passively tries to (i) infer clients' inputs, and (ii) link anonymous credentials to real identities.*

Definition 2 (Malicious client)**.** *A client may deviate arbitrarily, including (i) submitting constraint-violating inputs, (ii) encrypting with inconsistent keys to corrupt the aggregate, (iii) forging or refreshing anonymous credentials each round to evade the blacklist or impersonate others, and (iv) inferring honest clients' inputs or identities.*

If the server is malicious, it may deviate arbitrarily from the protocol, misclassifying honest clients as malicious—or vice versa. In SAIV schemes such as ACORN [17], a malicious server is obliged merely to identify malicious clients while still retaining their inputs, whereas WhiteCloak must additionally penalize misbehaving clients, making such punishment unavoidable. To counter this stronger adversary, we envisage a consensus list jointly maintained by the issuer, the server, and the clients. Every verification—including the proof, constraints, timestamp, and round identifier—is recorded, allowing clients to audit. The full design of this mechanism is beyond the present scope and is detailed in the appendix. The security requirements are as follows:

Definition 3 (Privacy)**.** *No server or client can extract private message of the client from the inputs.*

Definition 4 (Compliance)**.** *A client is only considered malicious if it submits inputs that not satisfy the constraints and encrypts them using the incorrect key.*

Definition 5 (Blacklistability)**.** *A client can only successfully authenticate to an honest server if it holds a valid credential that is not included in the blacklist.*

Definition 6 (Non-Frameability)**.** *It is not possible to prevent an honest client, who is not on the blacklist, from successfully authenticating with an honest server.*

Definition 7 (Anonymity)**.** *No server or client can distinguish between the authentication records of two honest clients, nor can they associate the authentication records with the clients' registration or identity information.*

III. WhiteCloak design and Overview

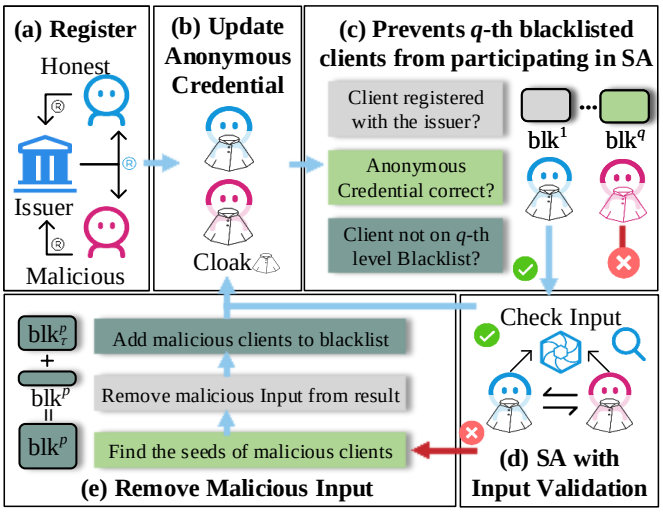

Fig. 1: (a) The client registers ID i with the issuer. (b) Updates anonymous credential. (c) The client generates three proofs to prove it is not on the blacklist. (d) Execute the SAIV to verify the input. (e) For a malicious client, add it to the p-th blacklist, and remove its input from the aggregation.

WhiteCloak introduces an innovative approach that enables clients to anonymously participate in SAIV while ensuring that malicious clients can be banned across rounds, as shown in Figure 1. WhiteCloak consists of 4 main components:

- **Client**: Each client has a unique ID i authenticated by the Issuer, which prevents Sybil attacks.The client generates an anonymous credential $\tilde{i}_\tau$ for round τ.
- **Issuer**: Issuer registers each client with a unique ID i and generates a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ using the $\mathsf{Schnorr.KeyGen}()$ protocol [50], [51]. If a single trusted issuer is unavailable, credentials can be issued collaboratively via distributed key generation and threshold signature schemes.
- **Server**: The server accepts inputs only from registered, non-blacklisted clients. It detects any malicious clients, who are then listed in blacklist and have their inputs removed.

System Initialization

1) Set $\tau = 0$ and initialize $\{\mathbf{blk}_\tau^z, \mathbf{blk}^z = \emptyset\}_{z=1}^q$.
2) Clients register with the issuer to generate the key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ using the Schnorr protocol. Additionally, the issuer signs a commitment $\mathsf{Com}(i, r_i)$ for ID i with Schnorr, producing σ_i^{id}, where r_i is a random value.
3) Each client i generates an anonymous credential $\tilde{i}_\tau = \mathsf{Prf}_i(\xi_\tau)$ using a randomly chosen value ξ_τ and demonstrates that its ID i is registered with the Issuer by computing $\pi_{\text{isu}} \leftarrow \text{REG.CLI.}(\mathsf{ck}, \sigma_i^{id}, r_i, i, R_{\text{isu}})$, where $(\mathsf{ck}, \mathsf{srs}) \leftarrow \mathsf{HICIAP.GenCk}(n)$, and n represents the number of proofs required for aggregation. Client i then participates in the aggregation process with $(\tilde{i}_\tau, \xi_\tau)$.

Forbid Malicious Clients:

4) The server broadcasts blacklists $\{\mathbf{blk}_\tau^z\}_{z=1}^q$ to all clients $\tilde{i}_\tau$, then updates $\tau \leftarrow \tau + 1$ and sets $\{\mathbf{blk}_\tau^z = \emptyset\}_{z=1}^q$.
5) If $\mathbf{blk}_{\tau-1}^q \neq \emptyset$, the server and clients update $(\mathsf{ck}, \mathsf{srs}) \leftarrow \mathsf{HICIAP.GenCk}(n)$. Each client i generates a new proof for the concealed common input i, demonstrating that client i has been registered by the Issuer with $\hat{\pi}_{\text{isu}} \leftarrow \text{REG.CLI.}(\mathsf{ck}, \sigma_i^{id}, r_i, i, R_{\text{isu}})$. Additionally, client i generates a proof for the new q-level blacklist $\mathbf{blk}_{\tau-1}^q$ and its concealed common input i's proof with $\pi_{\text{blk}}, \hat{\pi}_{\text{blk}} \leftarrow \text{BLK.PROOF}(\mathsf{ck}, i, \mathbf{blk}_\tau^q, q, R_{\text{blk}})$.
6) Client i updates its anonymous credential and proves that it was indeed generated from ID i by computing $(\tilde{i}_\tau, \xi_\tau), (\pi_{\text{anc}}, \hat{\pi}_{\text{anc}}) \leftarrow \text{AC.PROOF}(\mathsf{ck}, i, R_{\text{anc}})$, and links the proofs by $\pi_{\text{zkbl}} \leftarrow \text{LINKPROOF}(i, \{\hat{\pi}_{ope}\}, \{\pi_{ope}\})$, where $ope \in \{\text{isu, anc, blk}\}$. Client i then sends $(\pi_{\text{zkbl}}, \tilde{i}_\tau, \xi_\tau)$ to the server.
7) The server executes $\text{VERIFY}(\pi_{\text{zkbl}}, \xi_\tau, \tilde{i}_\tau, \mathsf{blk}_{\tau-1}^q, q, \mathsf{srs})$ for all clients $\tilde{i}_\tau$. If all verifications are successful, the server accepts client $\tilde{i}_\tau$ as not blacklisted; otherwise, participation is prohibited.

SA Participation of Verified Clients:

8) Clients and the server execute the SA protocol with input validation, such as described in [16], [17], [18], and utilize $\pi_{\mathbf{s}_{ij}}$ to obtain secret shares $H_{\tilde{i}_s}^s$, where $\tilde{i}_s$ refers to the anonymous credential used for seed generation. The aggregation result $\mathbf{a}$ is computed, and malicious clients $\{\tilde{m}_\tau\}$ are detected and temporarily added to a blacklist blk. If $\mathsf{blk} = \emptyset$, go to step 4) to start the next round of SA; Otherwise, go to step 9).

Remove Malicious Clients:

9) For each $\tilde{m}_\tau \in \mathsf{blk}$, identify the occurrence p at which its ID m was added to the blacklist, and subsequently update the set $\mathsf{blk}_\tau^p \leftarrow \mathsf{blk}_\tau^p \cup (\tilde{m}_\tau, \xi_\tau)$.
10) The anonymous credentials of the malicious clients $\tilde{m}_\tau \in \mathsf{blk}$ are identified for seed generation, which encompasses three scenarios to get remove list of secret key components R_{seed}. Then the server removes the inputs and secret key components associated with the malicious clients from round τ to get final result $\mathbf{a}^*$:

$$\mathsf{R}_{\text{seed}} = \begin{cases} \{\tilde{m}_\tau\} \in \mathsf{blk}, & \text{(single)} \\ \text{FINDSEED}^1(\mathsf{R}_{\text{seed}}, \tilde{i}_0, \tilde{i}_\tau, \mathsf{blk}), & \text{(multi)} \\ \text{FINDSEED}^2(\mathsf{R}_{\text{seed}}, \tilde{i}_0, \tilde{i}_\tau, \mathsf{blk}), & \text{(multi)} \end{cases}, \mathbf{a}^* = \begin{cases} \text{REMOVEINPUT}(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\text{seed}}, \cdot), & \text{(single)} \\ \text{REMOVEINPUT}(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\text{seed}}, \tau), & \text{(multi)} \end{cases}$$

11) Go to step 4) to start the next round of SA.

Fig. 2: WhiteCloak overview, focusing on its new features, excluding SAIV content.

- **Blacklist**: When a client is flagged as malicious p times, their anonymous credential is added to blk^p. Upon reaching blk^q, the client is barred from further participation. This design supports cross-round accountability without linking client identities to their banned status. It is maintained jointly by the server, the client and the Issuer.

In this approach, the system initializes with round $\tau = 0$ and an empty blacklist $\{\mathbf{blk}_\tau^z = \emptyset\}_{z=1}^q$. Clients register with the issuer, generating a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and a signature $\mathsf{Com}(i, r_i)$. The system then generates public strings, relationships, and other public parameters. Before each round, the server and clients update the round number τ, the blacklist $\{\mathbf{blk}_\tau^z\}_{z=1}^q$, and public parameters. Before participating in a new round of SA, each client generates a new anonymous credential $\tilde{i}_\tau$ and proves three things: 1) whether the client is a legitimate registered entity with the issuer; 2) whether the anonymous credential was generated using a registered ID; and 3) whether $\tilde{i}_\tau$ does not share the same ID as any credential in the blacklist blk. Only clients whose proofs pass all checks are allowed to proceed to the SA phase. The server then uses methods outlined in [16], [17], [18] to verify the SA inputs and identify any malicious clients. If any are detected, their anonymous credentials are added to the blacklist, preventing them from participating in future aggregation rounds. Furthermore, their inputs and associated keys are removed from the aggregation result. All of these actions are carried out while maintaining client anonymity. Despite clients' anonymity and message encryption, this method can

detect malicious submissions and exclude malicious clients from future participation, even if they attempt to rejoin using new anonymous credentials. WhiteCloak can directly utilize Mixnet [52] or Tor [53] technologies to conceal the IP address and ensure anonymity at the network layer.

The overview of WhiteCloak is shown in Figure 2 and the complete protocol is in the Appendix.

IV. Forbid Malicious Client in Anonymity

Because anonymous credentials are refreshed each round to break any link to real identities [54], tracking malicious clients across rounds is challenging. We solve this by adopting HICIAP [30], which hides Groth16's common reference string [29], and by introducing a hierarchical blacklist that stores each offender's pair $(\tilde{m}_\tau, \xi_{i,\tau})$ to reduce false positives. A client is admitted only when all three checks pass: (i) the ID is registered, (ii) the credential is bound to that ID, and (iii) the credential is not on the blacklist. This design preserves anonymity while enabling cross-round accountability.

A. Proof of Registration

In the anonymous credential system, clients register from an issuer that verifies their identity, protecting against Sybil attacks-that is, each client needs to have a unique immutable real identity. During registration, each client presents a unique ID i, which the issuer records within the relation R_{isu}:

$$R_{\text{isu}} := \left\{ (i, \mathsf{pk}_i, \sigma_i^{id}) \,\middle|\, \mathrm{Ver}_{\mathsf{pk}}(\mathsf{Com}(i, r_i), \sigma_i^{id}) \right\} \tag{5}$$

Eq. 5 guarantees that the client is using the ID it originally registered with the issuer, a prerequisite for both "**Non-Frameability**" and "**Anonymity**". It prevents malicious clients from forging credentials.

Clients register with the issuer to generate a key pair $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Schnorr.KeyGen}()$ and sign their ID i as $\sigma_i^{id} \leftarrow \mathsf{Schnorr.Sign}_{\mathsf{sk}_i}(\mathsf{com}_{i,r})$, where $r_i \leftarrow \{0,1\}^\lambda$ and $\mathsf{com}_{i,r} \leftarrow \mathsf{Com}(i, r_i)$, following the Schnorr protocol [50], [51]. The proof π_{isu} is generated by the function $\textsc{Reg.Cli.}(\mathsf{ck}, \sigma_i^{id}, r_i, i, R_{\text{isu}})$, which initiates the Groth16 setup and proving process, as illustrated in Figure 3. In this setup, the commitment key and short structured verification key are computed by $(\mathsf{ck}, \mathsf{srs}) \leftarrow \mathsf{HICIAP.GenCk}(n)$, supporting verification of HICIAP aggregates for up to $n-2$ Groth16 proofs. Utilizing HICIAP [30], this setup generates a concealed common input proof, $\hat{\pi}_{\text{isu}}$, thereby ensuring that the client's ID i is both verified and securely concealed. This is a precondition to guarantee "**Anonymity**" and prevent servers or clients from inferring or linking the true identities of other clients.

B. Proof of Bound of Anonymous Credential

Once client i completes registration with the issuer, it can generate an anonymous credential $\tilde{i}_\tau = \mathsf{Prf}_i(\xi_{i,\tau})$ using its identity i and a randomly generated value $\xi_{i,\tau} \leftarrow \{0,1\}^\lambda$ for each round, as shown in relation R_{anc}:

$$R_{\text{anc}} := \left\{ (i, \tilde{i}_\tau, \xi_{i,\tau}) \,\middle|\, \mathsf{Prf}_i(\xi_{i,\tau}) = \tilde{i}_\tau \right\}. \tag{6}$$

1) $\textsc{Reg.Cli.}(\mathsf{ck}, \sigma_i^{id}, r_i, i, R_{\text{isu}}) \rightarrow (\pi_{\text{isu}}, \hat{\pi}_{\text{isu}})$
   $\mathrm{crs}_{\text{isu}} \leftarrow \mathsf{Groth16.Setup}(R_{\text{isu}})$
   $\pi_{\text{isu}} \leftarrow \mathsf{Groth16.Prove}(\mathrm{crs}_{\text{isu}}; i; \sigma_i^{id}, r_i)$
   $\hat{\pi}_{\text{isu}} \leftarrow \mathsf{HICIAP.Prove}(\mathsf{ck}; \mathrm{crs}_{\text{isu}}; i, \pi_{\text{isu}})$
2) AC. $\textsc{Proof}(\mathsf{ck}, i, R_{\text{anc}}) \rightarrow (\tilde{i}_\tau, \xi_{i,\tau}), (\pi_{\text{anc}} \hat{\pi}_{\text{anc}})$
   $\mathrm{crs}_{\text{anc}} \leftarrow \mathsf{Groth16.Setup}(R_{\text{anc}})$
   $\xi_{i,\tau} \leftarrow \{0,1\}^\lambda$, $\tilde{i}_\tau \leftarrow \mathsf{Prf}_i(\xi_{i,\tau})$
   $\pi_{\text{anc}} \leftarrow \mathsf{Groth16.Prove}(\mathrm{crs}_{\text{anc}}; i, \tilde{i}_\tau, \xi_{i,\tau}; \cdot)$
   $\hat{\pi}_{\text{anc}} \leftarrow \mathsf{HICIAP.Prove}(\mathsf{ck}; \mathrm{crs}_{\text{anc}}; \tilde{i}_\tau, \xi_{i,\tau}; i, \pi_{\text{anc}})$
3) $\textsc{Blk.Proof}(\mathsf{ck}, i, \mathsf{blk}_\tau^p, p, R_{\text{blk}}) \rightarrow (\pi_{\text{blk}}, \hat{\pi}_{\text{blk}})$
   $\mathrm{crs}_{\text{blk}} \leftarrow \mathsf{Groth16.Setup}(R_{\text{blk}})$
   $\pi_{\mathsf{blk}_\tau^p} \leftarrow \mathsf{Groth16.Prove}(\mathrm{crs}_{\text{blk}}; i, \mathsf{blk}_\tau^p; \cdot)$
   $\hat{\pi}_{\mathsf{blk}^p} \leftarrow \mathsf{HICIAP.Prove}(\mathsf{ck}, \mathrm{crs}_{\text{blk}}; \cdot; i, \{\pi_{\mathsf{blk}_z^p}\}_{z=1}^{\tau})$
4) $\textsc{LinkProof}(i, \{\hat{\pi}_{ope}\}) \rightarrow \pi_{\text{zkbl}}$
   $\pi_{\text{link}} \leftarrow \mathsf{HICIAP.LinkProve}(i; \{\hat{\pi}_{ope}\}; i)$
   $\pi_{\text{zkbl}} \leftarrow (\pi_{\text{link}}, \{\hat{\pi}_{ope}\})$
5) $\textsc{Verify}(\mathsf{ck}, \pi_{\text{zkbl}}, \xi_{i,\tau}, \tilde{i}_\tau, \mathsf{blk}_\tau^p, p, \mathsf{srs}) \rightarrow \mathsf{Result}$
   $\hat{\mathbf{S}}_{\mathsf{blk}_\tau^p} \leftarrow \mathsf{Groth16.Prepare}(\mathrm{crs}_{\text{blk}}; \mathsf{blk}_\tau^p)$
   $\hat{\mathbf{S}}_{\mathsf{blk}^p} \leftarrow \mathsf{HICIAP.Com}(\mathsf{ck}, \{\hat{\mathbf{S}}_{\mathsf{blk}_z^p}\}_{z=1}^{\tau})$
   $\hat{\mathbf{S}}_{\text{isu}} \leftarrow \mathsf{Groth16.Prepare}(\mathrm{crs}_{\text{isu}}; \cdot)$
   $\hat{\mathbf{S}}_{\text{anc}} \leftarrow \mathsf{Groth16.Prepare}(\mathrm{crs}_{\text{anc}}; \tilde{i}_\tau, \xi_{i,\tau})$
   $\mathsf{Result} \leftarrow \mathsf{HICIAP.LinkVfy}(\pi_{\text{link}}; \hat{\pi}_{\text{isu}}, \hat{\pi}_{\text{anc}}, \hat{\pi}_{\mathsf{blk}^p})$
   $\bigwedge_{ope \in \{\text{isu,anc,blk}^p\}} \mathsf{HICIAP.Vfy}(\mathsf{srs}; \hat{\pi}_{ope}; \hat{\mathbf{S}}_{ope})$

Fig. 3: Functions in Blacklist of WhiteCloak

Here, τ represents the current round, and $\mathsf{Prf}(\cdot)$ denotes a pseudorandom function [55]. Eq. 6 guarantees the binding between $\tilde{i}_\tau$ and i, a prerequisite for both "**Non-Frameability**" and "**Anonymity**", and prevents malicious clients from forging credentials or impersonating others.

The proof π_{anc} and the concealed input proof $\hat{\pi}_{\text{anc}}$ are generated by the function $\text{AC.}\textsc{Proof}(\mathsf{ck}, i, R_{\text{anc}})$ as illustrated in Figure 3, which constructs the anonymous credential $\tilde{i}_\tau$ along with a random value $\xi_{i,\tau}$ for the round τ. It then creates a Groth16 proof related to the anonymous relation R_{anc} and conceals the common input in $\hat{\pi}_{\text{anc}}$. π_{anc} is used to prove that the anonymous credential $\tilde{i}_\tau$ was indeed generated using i. This is a precondition to guarantee "**Anonymity**" and prevent servers or clients from inferring or linking the true identities of other clients.

C. Proof of Non-Blacklist

To implement accountability while preserving anonymity and misjudgment tolerance, we designed a multi-level anonymous blacklist. This blacklist allows clients some leniency in cases of incidental errors, permitting up to q infractions before permanent blacklisting, thus balancing accountability with tolerance. The multi-level anonymous blacklist comprises q levels $\{\mathbf{blk}^p\}_{p=1}^q$. Before each aggregation round τ, every client must demonstrate that this credential does not appear in the q-th blacklist $\mathbf{blk}^q$, thereby fulfilling the relation R_{blk}:

$$R_{\text{blk}} := \left\{ (i, \mathsf{blk}^q) \,\middle|\, \forall (\tilde{m}_\tau, \xi_{i,\tau}) \in \mathsf{blk}^q, \mathrm{Prf}_i(\xi_{i,\tau}) \neq \tilde{i}_\tau \right\} \tag{7}$$

Eq. 7 guarantees that the identity i responsible for $\tilde{i}_\tau$ has no anonymous credential recorded in $\mathbf{blk}^q$, thereby upholding both "**Blacklistability**" and "**Non-Frameability**".

If the condition is met, the client is allowed to participate in aggregation. Blacklist verification is based on proofs $\pi_{\mathsf{blk}^q_{\tau-1}}$ and concealed input proofs $\hat{\pi}_{\mathsf{blk}^q_\tau}$ generated by the BLK.PROOF$(\mathsf{ck}, i, \mathsf{blk}^q_{\tau-1}, q, R_{\text{blk}})$ function as illustrated in Figure 3, here, $\mathbf{blk}^q_{\tau-1}$ represents the q-th level blacklist that is added in round τ. This function uses HICIAP to create a Groth16 proof $\pi_{\mathsf{blk}^q_{\tau-1}}$ for the blacklist set $\mathsf{blk}^q_{\tau-1}$, effectively concealing the common input and ensuring that clients can prove they are not blacklisted without revealing their identity. This is a precondition to guarantee "**Anonymity**" and prevent servers or clients from inferring or linking the true identities of other clients.

The blacklist $\mathbf{blk}^q$ is composed of $\{\mathbf{blk}^q_1, \ldots, \mathbf{blk}^q_{\tau-1}\}$. Whenever a client's anonymous credential $\tilde{i}_\tau$ is detected for the q-th infraction in round τ, the credential and its random value pair $(\tilde{i}_\tau, \xi_{i,\tau})$ are added to that round's blacklist blk^q_τ, forming part of the q-level blacklist $\mathbf{blk}^q$. In round τ, client i only needs to prove that their ID does not appear in the latest generated blacklist $\mathbf{blk}^q_{\tau-1}$. For previous round blacklists $\{\mathbf{blk}^q_1, \ldots, \mathbf{blk}^q_{\tau-2}\}$, the client can reuse their proof from the last round, avoiding redundant verification.

D. Hide the true identity and aggregate the three proofs

We have now generated proofs associated with R_{isu}, R_{anc}, and R_{blk}. However, these proofs need to use the same i, which requires linking them together. By utilizing HICIAP [30], we can effectively conceal the shared input i within the underlying Groth16 proofs [29], generating implicit proofs $\hat{\pi}_{\text{isu}}$, $\hat{\pi}_{\text{anc}}$, and $\hat{\pi}_{\text{blk}}$. The aggregated multi-ZKP framework streamlines the multi-stage verification process and handles private inputs efficiently. Without aggregation, separate Σ-protocol proofs [56] would need additional consistency checks to ensure every instance references the same i. Leveraging aggregation, we define the relation R_{zkbl} (Eq. 8), where (i, σ_i^{id}, r_i) are private inputs and $(\mathsf{blk}_z)_{z=1}^\tau, \mathsf{pk}_i, \tilde{i}_\tau, \xi_{i,\tau}$ are public. A valid proof must satisfy R_{isu}, R_{anc}, and R_{blk} simultaneously under the same i. Concealing i yields **Anonymity**, while the three combined constraints ensure both **Blacklistability** and **Non-Frameability**. Thus, the system blocks malicious clients who forge or refresh credentials to evade the blacklist or impersonate others, and prevents the server or peers from inferring honest clients' identities.

$$R_{\text{zkbl}} := \left\{ \begin{array}{l} ((\mathsf{blk}_z)_{z=1}^\tau, \mathsf{pk}_i, \tilde{i}_\tau, \xi_{i,\tau}; i, \sigma_i^{id}, r_i) \mid \\ R_{\text{anc}}(i, \tilde{i}_\tau, \xi_{i,\tau}) \bigwedge_{z=1}^\tau R_{\text{blk}}(i, \mathsf{blk}_z) \\ \wedge R_{\text{isu}}(i, \mathsf{pk}_i, \sigma_i^{id}, r_i) \end{array} \right\} \quad (8)$$

The proof $\pi_{\text{zkbl}} \leftarrow$ LINKPROOF$(i, \hat{\pi}_{\text{isu}}, \hat{\pi}_{\text{anc}}, \hat{\pi}_{\text{blk}})$ function, which links the various proofs in Figure 3. Using HICIAP, this function generates π_{link}, merging multiple proofs into π_{zkbl}, thus enabling comprehensive anonymous blacklist verification.

The server uses the VERIFY$(\mathsf{ck}, \pi_{\text{zkbl}}, \xi_{i,\tau}, \tilde{i}_\tau, \mathsf{blk}^q_{\tau-1}, q, \mathsf{srs})$ function to determine whether the client $\tilde{i}_\tau$ appears in the q-th level blacklist, verifying the validity of the aggregated proof π_{zkbl}. See the Appendix B for instantiation.

E. Tolerance for up to q Infractions

When an anonymous client $\tilde{i}_\tau$ is flagged as malicious, the system must learn how many previous infractions it has accrued. If this is the p-th incident, the pair $(\tilde{i}_\tau, \xi_\tau)$ already appears in $\mathbf{blk}^1_\tau$ through $\mathbf{blk}^{p-1}_\tau$. Starting the search at level $q-1$ locates the pair in $O(q-p)$ time with a linear scan or $O(\log q)$ with binary search. If found, it is promoted to $\mathbf{blk}^p_\tau$; otherwise it enters $\mathbf{blk}^1_\tau$. Efficiency improves if the client discloses its current highest level p. The verifier then inspects only levels p and $p+1$: absence from level $p+1$ places the pair there, while presence sends it directly to level q. This constant-time update is certified by R_{zkbl}, which subsumes the constraints R_{anc} and R_{isu}. Tune the tolerance dynamically via

$$q \leftarrow q + \alpha\big(FP_{\text{rate}} - T_{FP}\big),$$

where FP_{rate} is the observed false-positive rate, T_{FP} the target rate, and α the adjustment step.

V. REMOVE MALICIOUS CLIENT IN ANONYMITY

When a client $\tilde{i}_\tau$ is blacklisted, its encrypted input $\mathbf{y}_{\tilde{i}_\tau}$ must be removed, while prior validated inputs remain safe for SA. Direct deletion would break Eq. 2, since every pairwise key $\mathbf{s}_{\tilde{i}_\tau \tilde{j}_\tau}$ must remain. Honest clients therefore use Shamir recovery [48] to rebuild $\text{seed}_{\tilde{i}_\tau \tilde{j}_\tau}$ and derive $\mathbf{s}_{\tilde{i}_\tau \tilde{j}_\tau} \leftarrow \mathsf{Prf}_i(\text{seed}_{\tilde{i}_\tau \tilde{j}_\tau})$. With these keys restored, the server can safely strip $\mathbf{y}_{\tilde{i}_\tau}$ without exposing the client's identity. This ensures that WhiteCloak can remove malicious clients that submitting constraint-violating inputs and keys.

A. One Seed Within One Interaction Round

In the SA protocol proposed by Bell et al. [17], [31], each client uses a unique random seed $\mathbf{seed}_{ij}$ in each round, which requires clients to generate a new random seed via key exchange for every round of SA. In this context, even in an anonymous setting, it remains feasible to directly remove the inputs and associated keys of a malicious client. Specifically, in the τ-th round, a malicious client m participates in the aggregation with an anonymous credential $\tilde{m}_\tau$ and generates random seeds $\{\mathbf{seed}_{\tilde{m}_\tau \tilde{j}_\tau}\}$ with its neighboring clients set $\mathcal{A}_{\tilde{m}_\tau}$, where $\tilde{j}_\tau \in \mathcal{A}_{\tilde{m}_\tau}$. This setup allows the server and honest clients to directly remove the associated keys $\mathbf{s}_{\tilde{m}_\tau \tilde{j}_\tau}$ and the input $\mathbf{y}_{\tilde{m}_\tau}$ based on the anonymous credential $\tilde{m}_\tau$. First, the anonymous credential $\tilde{m}_\tau$ of the malicious client is extracted from the newly generated blacklist blk for this round. This credential, used to generate random seeds, is then added to the set of seeds to be removed, R_{seed}. Next, the function REMOVEINPUT$(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\text{seed}}, \cdot)$ is executed, as shown in Figure 4, to remove the input $\mathbf{y}_{\tilde{m}_\tau}$ and its associated key component $\mathbf{s}_{\tilde{m}_\tau \tilde{j}_\tau}$ from the aggregation result $\mathbf{a}$. Here $\mathcal{A}_{\tilde{i}_s}$ is the set of verified neighbors of $\tilde{i}_s$, $H^s_{\tilde{i}_s}$ representing the secret share. Because the entire process reveals neither the malicious client's identity, message, nor keys—and removes all non-compliant term—it preserves **"Anonymity"** while maintaining both **"Compliance"** and **"Privacy"**.

1) $\textsc{RemoveInput}(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\text{seed}}, \tau) \to \mathbf{a}^*$
$\mathsf{sk}_{\tilde{i}_s} = \mathsf{Recover}(H^s_{\tilde{i}_s})$
$\mathbf{seed}_{\tilde{i}_s\tilde{j}_s} = \mathsf{KA.Agree}(\mathsf{sk}_{\tilde{i}_s,1}, \mathsf{pk}_{\tilde{j}_s,1})$
$\mathbf{s}_{\tilde{i}_s\tilde{j}_s} = \delta_{\tilde{i}_s\tilde{j}_s}\mathsf{Prf}(\mathbf{seed}_{\tilde{i}_s\tilde{j}_s}, \tau)$.
$\mathbf{a}^* = \mathbf{a} - \sum_{\tilde{m}\in\mathsf{blk}} \mathbf{y}_{\tilde{m}} + \sum_{\tilde{i}_s\in\mathsf{R}_{\text{seed}}} \sum_{\tilde{j}_s\in\mathcal{A}_{\tilde{i}_s}} \delta_{\tilde{i}_s\tilde{j}_s}\mathbf{s}_{\tilde{i}_s\tilde{j}_s}$
2) $\textsc{FindSeed}^1(\mathsf{R}_{\text{seed}}, \tilde{i}_0, \tilde{i}_\tau, \mathbf{blk}) \to \mathsf{R}_{\text{seed}}$
$\mathsf{crs}_{\text{sim}} \leftarrow \mathsf{Groth16.Setup}(R_{\text{sim}}(i, (\tilde{i}_k, \xi_{i,k})_{k\in\{\tau,0\}}))$
$\pi_{\text{sim}} \leftarrow \mathsf{Groth16.Prove}(\mathsf{crs}_{\text{sim}}; \tilde{i}_0, \tilde{i}_\tau; i)$.
$\mathsf{Groth16.Vfy}(\mathsf{crs}_{\text{sim}}; \pi_{\text{sim}}; \tilde{i}_0, \tilde{i}_\tau) = \mathsf{false} \vee \tilde{i}_\tau \in \mathbf{blk}$
$\implies \mathsf{R}_{\text{seed}} \leftarrow \mathsf{R}_{\text{seed}} \cup \{\tilde{i}_0\}$.
3) $\textsc{FindSeed}^2(\mathsf{R}_{\text{seed}}, \tilde{i}_0, \tilde{i}_\tau, \mathbf{blk}) \to \mathsf{R}_{\text{seed}}$
$(\tilde{i}_{\tilde{\tau}}, \xi_{\tilde{\tau}}), (\hat{\pi}_{\text{anc}}, \pi_{\text{anc}}) \leftarrow \textsc{AC.Proof}(\mathsf{ck}, i, R_{\text{anc}})$;
$(\hat{\pi}_{\text{blk}}, \pi_{\text{blk}}) \leftarrow \textsc{Blk.Proof}(\mathsf{ck}, i, \mathbf{blk}, \cdot, R_{\text{blk}})$;
$\mathsf{crs}_{\text{sim}} \leftarrow \mathsf{Groth16.Setup}(R_{\text{sim}}(i, (\tilde{i}_k, \xi_{i,k})_{k\in\{\tilde{\tau},0\}}))$;
$\pi_{\text{sim}} \leftarrow \mathsf{Groth16.Prove}(\mathsf{crs}_{\text{sim}}; \tilde{i}_0, \tilde{i}_{\tilde{\tau}}, i; \cdot)$;
$\hat{\pi}_{\text{sim}} \leftarrow \mathsf{HICIAP.Prove}(\mathsf{ck}, \mathsf{crs}_{\text{sim}}; \tilde{i}_0, \tilde{i}_{\tilde{\tau}}; i, \pi_{\text{sim}})$;
$\pi_{\text{zkbl}} \leftarrow \textsc{LinkProve}(i; \{\pi_o\}; \{\hat{\pi}_o\})_{o\in\{\text{isu,anc,blk,sim}\}}$.
$\textsc{Verify}(\mathsf{ck}, \pi_{\text{zkbl}}, \xi_{\tilde{\tau}}, \tilde{i}_{\tilde{\tau}}, \mathsf{blk}, \cdot, \mathsf{srs}) = \mathsf{false}$
$\implies \mathsf{R}_{\text{seed}} \leftarrow \mathsf{R}_{\text{seed}} \cup \{\tilde{i}_0\}$;

Fig. 4: Functions of remove Malicious clients.

B. One Seed In Multiple Interaction Rounds:

Clients reuse the same random seed $\mathbf{seed}_{ij}$ to generate keys across multiple rounds of SA, such as Flamingo [33]. We assume this random seed, $\mathbf{seed}_{\tilde{i}_0\tilde{j}_0}$, was initially generated in round $\tau = 0$ through key exchange. For the τ-th round of aggregation, the anonymous credential associated with the client's input is $\tilde{i}_\tau$, while the credential used for key generation remains $\tilde{i}_0$. The generated key is:

$$\mathbf{s}^{\tau}_{\tilde{i}_0\tilde{j}_0} = \delta_{\tilde{i}_0\tilde{j}_0}\mathsf{Prf}(\mathbf{seed}_{\tilde{i}_0\tilde{j}_0}, \tau), \quad \delta_{\tilde{i}_0\tilde{j}_0} = \begin{cases} 1, & \text{if } \tilde{i}_0 > \tilde{j}_0 \\ -1, & \text{if } \tilde{i}_0 < \tilde{j}_0. \end{cases}$$

To remove the input and key of a malicious client $\tilde{i}_\tau$, the "**unlinkability**" of the anonymous prevents the server and other clients from associating $\tilde{i}_\tau$ with $\tilde{i}_0$. As a result, the key $\mathbf{s}^{\tau}_{\tilde{i}_0\tilde{j}_0}$ cannot be removed as directly. To address this, it is necessary to demonstrate that they share the same underlying ID i, as specified by the relation R_{sim}:

$$R_{\text{sim}} := \left\{(i, (\tilde{i}_k, \xi_{i,k})_{k\in\{\tau,0\}}) \mid \forall k \in \{\tau, 0\},\, \mathsf{Prf}_i(\xi_{i,k}) = \tilde{i}_k\right\}. \tag{9}$$

This relationship ensures that honest clients do not satisfy the constraints, but malicious clients will. Therefore, the relationship R_{Fin_1} can be constructed:

$$R_{\text{Fin}_1} := \left\{ \begin{array}{l} (i, \mathsf{blk}, (\tilde{k}, \xi_{i,k})_{k\in\{\tau,0\}}) \mid \\ (\tilde{i}_\tau, \xi_\tau) \notin \mathsf{blk} \wedge R_{\text{sim}}(i, (\tilde{i}_k, \xi_{i,k})_{k\in\{\tau,0\}}) \end{array} \right\} \tag{10}$$

Eq. 10 indicates that for any client that meets the condition R_{sim}, it is not on the blacklist. Therefore, any client that fails to provide this relation or whose verification fails will be removed from the aggregation and its input and key need to be removed. Consequently, the threat of malicious clients submitting constraint-violating messages and keys is eliminated. The key step here is to identify the set of seeds that need to be removed, R_{seed}, which can be accomplished through the function $\textsc{FindSeed}^1(\mathsf{R}_{\text{seed}}, \tilde{i}_0, \tilde{i}_\tau, \mathsf{blk})$ as shown in Figure 4. First, all parties use Groth16 to generate the CRS for R_{sim}, and each client generates a proof π_{sim}. The server then includes all $\tilde{m}_0$ credentials that did not pass verification into R_{seed}. The function $\textsc{RemoveInput}(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\text{seed}}, \tau)$ is then applied to remove the inputs and keys of malicious clients from the aggregation result $\mathbf{a}$, yielding $\mathbf{a}^*$.

However, this approach presents a significant privacy concern in linking $\tilde{i}_\tau$ with $\tilde{i}_0$. If $\tilde{i}_0$ continues to be used for key generation, it may compromise privacy by allowing adversaries to establish correlations across rounds, thereby breaking unlinkability. Therefore, **after executing $\textsc{FindSeed}^1$, it is essential to use a new anonymous credential to generate a fresh random seed.**

For scenarios where the cost of replacing random seeds is high, we propose a variant scheme. Each client i generates a new anonymous credential $\tilde{i}_{\tilde{\tau}}$ in round τ when removing a malicious client is required. Client i proves that $\tilde{i}_{\tilde{\tau}}$ is not in the blacklist blk and further associates $\tilde{i}_{\tilde{\tau}}$ with $\tilde{i}_0$ to verify that they correspond to the same identity i. This method circumvents the need to directly link $\tilde{i}_\tau$ and $\tilde{i}_0$. The client uses $\tilde{i}_\tau$ to send messages, while $\tilde{i}_{\tilde{\tau}}$ serves solely to prove non-membership in the blacklist. Consequently, this ensures the unlinkability between $\tilde{i}_\tau$ and $\tilde{i}_0$. Thus, the required relation is R_{Fin_2}:

$$R_{\text{Fin}_2} := \left\{ \begin{array}{l} (i, \mathsf{blk}, (\tilde{k}, \xi_{i,k})_{k\in\{\tilde{\tau},0\}}) \mid \bigwedge_{p=1}^{q} R_{\text{blk}}(i, \mathsf{blk}^p_\tau) \\ \wedge R_{\text{isu}}(i, \mathsf{pk}_i, \delta^{id}_i, r_i) \wedge R_{\text{anc}}(i, \tilde{i}_{\tilde{\tau}}, \xi_{\tilde{\tau}}) \\ \wedge R_{\text{sim}}(i, (\tilde{i}_k, \xi_{i,k})_{k\in\{\tilde{\tau},0\}}) \end{array} \right\} \tag{11}$$

Eq. 11 shows that $\tilde{i}_{\tilde{\tau}}$ must satisfy the constraints of R_{blk}, R_{isu}, R_{anc}, R_{sim}—i.e., $\tilde{i}_{\tilde{\tau}}$ must not be on the blacklist, its ID must be registered, it must be generated from a registered ID, and it must share the same ID as $\tilde{i}_0$. This relationship is a precondition to guarantee "**Anonymity**" while also ensures "**Blacklistability**" and "**Non-Frameability**", prevents malicious clients from bypassing the blacklist by forging or updating an anonymous credential.

R_{seed} can be obtained through $\textsc{FindSeed}^2(\mathsf{R}_{\text{seed}}, \tilde{i}_0, \tilde{i}_\tau, \mathsf{blk})$, as shown in Figure 4. $\textsc{RemoveInput}(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\text{seed}}, \tau)$ is then called to obtain the final aggregation result $\mathbf{a}^*$.

The functions $\textsc{FindSeed}^1$ and $\textsc{FindSeed}^2$ both rely on Groth16 proofs, with the key difference being that $\textsc{FindSeed}^1$ only requires proving the relation R_{Fin_1}, where i is treated as a private input. In contrast, in R_{Fin_2}, i is considered a public input within Groth16. Therefore, in addition to proving $R_{\text{sim}}, R_{\text{anc}}, R_{\text{isu}}, R_{\text{blk}}$, HICIAP is used to conceal the public input i.

The $\textsc{RemoveInput}$ only removes the malicious client (that is, in line with existing ACORN and RoFL detection standards), not for any client outages. The tolerance for dropped clients remains unchanged, and the protocol retains the original SAIV dropped tolerance feature. If the server is malicious, this

protocol will not be able to safely remove the client input, only the malicious client flag, not the RemoveInput action. Therefore, this extended security assumption is consistent with the original SAIV drop tolerance feature and poses no additional risk.

VI. Identify Malicious Clients with Incorrect Key

SAIV methods like ACORN [17] and RoFL [16] can identify clients submitting incorrect inputs but fail to detect those using incorrect keys. Thus, the proposed method can be applied after key verification in RoFL or ACORN to identify clients with incorrect keys.

Detection of clients using incorrect keys can be achieved by verifying constraints on individual client keys, specifically using the relation defined by Equation 1. Based on these constraints, we construct the relation $\mathsf{R}_{s_{ij}}$, where $\{\mathbf{c}_{ij}\}_j$ represents the Pedersen commitments for each $\{\mathbf{s}_{ij}\}_j$:

$$\mathsf{R}_{s_{ij}} = \left\{ \begin{array}{l} (\mathbf{y}_i, \mathbf{r}_i, \{\mathbf{c}_{ij}\}_j); (\mathbf{m}_i, \mathbf{s}_i, \{\mathbf{s}_{ij}\}_j) : \\ \mathbf{c}_{ij} = \mathsf{Com}(0, \mathbf{s}_{ij}) \wedge \mathbf{y}_i = \mathsf{Com}(\mathbf{m}_i, \mathbf{s}_i) \\ \wedge \mathbf{s}_i = \mathbf{r}_i + \sum_j \delta_{ij} \mathbf{s}_{ij} \end{array} \right\} \quad (12)$$

Eq. 12 ensures that the $\mathbf{c}_{ij}$, $\mathbf{y}_i$, and $\mathbf{s}_i$ share a consistent $\mathbf{s}_{ij}$, thus guaranteeing "**Compliance**" and preventing malicious clients from submitting keys that violate the constraints. This process ensures "**Privacy**" by preventing both clients and servers from stealing keys and messages.

Here, we use pedersen commitment [49] because both RoFL [16] and ACORN [17], in their architectures, use commitment based on discrete logarithms for zero-knowledge proofs or cryptograph commitments on inputs. In this way, more well-formedness proofs can be avoided. A dedicated ZKP protocol can be constructed based on this relation for verification. The protocol process is shown in Figure 5. This ZKP ensures the validity of the relationships among $\mathbf{s}_{ij}$, $\mathbf{s}_i$, and $\mathbf{r}_i$, but does not guarantee the correctness of each $\mathbf{s}_{ij}$ individually. Therefore, additional verification of each $\mathbf{s}_{ij}$ is required. Since $\mathbf{s}_{ij}$ is generated by $\mathsf{Prf}(\mathbf{seed}_{ij})$, it should remain consistent between clients i and j. The verifier can compare the commitment $\mathsf{Com}(0, \mathbf{s}_{ij})$ sent by client i with $\mathsf{Com}(0, \mathbf{s}_{ji})$ sent by client j. If they match, then $\mathbf{s}_{ij}$ and $\mathbf{s}_{ji}$ are consistent; otherwise, it indicates that at least one client is dishonest. In such cases, both clients i and j are required to reveal $\mathbf{s}_{ij}$ and $\mathbf{s}_{ji}$. Other clients can then use secret sharing to reconstruct $\mathbf{s}_{ij}$ (similar to how SA recovers data from a disconnected client) and compare it with the revealed values to identify the malicious client. Even if clients i and j collude, their shared value $\mathbf{s}_{ij}$ will still cancel out during aggregation, rendering collusion attempts ineffective.

By executing the ZKP of the relation $\mathsf{R}_{s_{ij}}$, we can effectively identify malicious clients using incorrect keys and add them to the blacklist blk.

1) The prover randomly generates k sets of random values $\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j \leftarrow \mathbb{Z}_q^\ell$, where k represents the number of neighboring clients with whom they jointly generate keys. Using Eq. 3, the prover then generates commitments $\{\mathbf{t}_j = \mathsf{Com}(\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j), \mathbf{c}_{ij} = \mathsf{Com}(0, \mathbf{s}_{ij})\}_j$. Note that $\mathbf{y}_i$ is already used for SA aggregation, so it does not need to be resent.
2) The verifier generates two random challenges c, b from the challenge domain and sends them to the prover. This process can also be made non-interactive using the Fiat-Shamir heuristic [57], where c, b are computed as the hash of $\mathbf{c}_{ij}, \mathbf{t}_{ij}, \mathbf{c}_i$.
3) The prover then computes responses using the challenges c, b and their witness, $(\mathbf{m}_i, \mathbf{s}_i, \{\mathbf{s}_{ij}\}_j)$, as follows:

$$\mathbf{R}_m = c\mathbf{m}_i + \textstyle\sum_j \delta_{ij} \boldsymbol{\alpha}_j, \mathbf{R}_s = (c+b)\mathbf{s}_i + \sum_j \delta_{ij} \boldsymbol{\beta}.$$

The prover then sends these responses to the verifier.
4)The verifier checks whether the following equality holds:

$$\mathsf{Com}(\mathbf{R}_m, \mathbf{R}_s) = \mathsf{Com}(\mathbf{x}_i, \mathbf{s}_i)^c \mathsf{Com}(0, \mathbf{r}_i)^b \cdot \prod_j \mathsf{Com}(\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j)^{\delta_{ij}} \mathsf{Com}(0, \mathbf{s}_{ij})^{\delta_{ij} b}.$$

If it does not hold, then the client key must be wrong.

Fig. 5: **Protocol** $\pi_{s_{ij}}$, Client Key Correctness Proof

VII. Security Analysis

Our security evaluation builds on the SAIV's security foundations, such as ACORN and RoFL. We do not repeat the discussions on **privacy** and **compliance** which are provided by SAIV. We focus on three key aspects introduced by WhiteCloak. The following cryptographic assumptions hold:

Assumption 1. $\mathsf{Groth16}$ *and* HICIAP *proofs are knowledge-sound and subversion zero-knowledge.* $\mathsf{Schnorr}$ *signatures are unforgeable [50], [51].* Prf *is pseudorandom [55]. Pedersen Commitment* Com *[49] is both binding and hiding.*

A. Forbid and remove Malicious

Theorem 1 (Forbid Malicious Security)**.** *The "Forbid Malicious Clients" and "Remove Malicious Clients" protocols, as described in Figures 2, 3, and Sections IV and V, ensure blacklistability, non-frameability and anonymity under the cryptographic assumptions specified in Assumption 1.*

Proof. We use the same proof as SNARKBlock [30].

Blacklistabilty. Let A be an adversary that breaks **Blacklistability**. Our goal is to prove that if A can bypass the blacklist mechanism, then Assumption 1 will be violated. For A to forge a verifiable proof $(\pi_{\text{zkb}}, \tilde{i}_\tau, \xi_{i,\tau})$, we consider the following scenarios:

1) A forges an anonymous credential such that $\tilde{i}_\tau = \mathsf{Prf}_{i'}(\xi_{i,\tau})$, where $i' \neq i$, which breaks the randomness of Prf.

2) A forges a HICIAP proof, and the proof does not use the corresponding Groth16 proof and the real identity i, thereby breaking the knowledge soundness of HICIAP.
3) A forges a Groth16 proof, and the content of the proof differs from the real content, which breaks the knowledge soundness of Groth16.
4) A forges a signature $\sigma_i^{id'}$, which is verified by pk_i and corresponds to $\mathsf{Com}(i, r_i)$. This breaks the unforgeability of the Schnorr signature.
5) A forges a HICIAP.Link link proof and successfully verifies it, which directly breaks the knowledge soundness of HICIAP.Link.
6) A finds a pair $(i', r_i') \neq (i, r_i)$, such that $\mathsf{Com}(i', r_i') = \mathsf{Com}(i, r_i)$, directly breaking the binding property of Peterson commitment Com.

In conclusion, the security of the components used by each function in Figure 3, 4 has been proven, ensuring that clients cannot forge identities or proofs to bypass the blacklist verification. Moreover, the server is secure under the semi-honest assumption, ensuring it does not deviate from the protocol. □

Non-Frameability. Let A be an adversary attempting to prevent an honest, non-blacklisted client from successfully authenticating.

If A successfully forges a valid pair $(\tilde{i}_\tau', \xi_{i,\tau}')$, it could cause the honest client to be mistakenly added to the blacklist. However, since Prf is pseudorandom, without knowing the true identity i, the adversary cannot guess the correct anonymous credential Therefore, A cannot forge a valid pair$(\tilde{i}_\tau', \xi_{i,\tau}')$. Furthermore, the anonymity guarantees that the adversary cannot distinguish between the authentication records of different honest clients, nor can they deduce a client's identity from the anonymous credential. Thus, the adversary cannot bypass the authentication process by generating a fraudulent tag.

Since the adversary cannot forge a valid anonymous credential or circumvent the authentication, WhiteCloak satisfies the non-frameability property. □

Anonymity. Based on the hiding property of the commitment scheme Com, the commitment com reveals no information about the identity i. Additionally, random value r_i is used during the registration process, ensuring that the commitment does not expose any sensitive information about the user. Since HICIAP is a zero-knowledge protocol, the proof π_{zkb} does not leak any information regarding the user's identity or registration details to the server. The nonce $\xi_{i,\tau}$ used in each session is chosen uniformly at random, ensuring that no information about the identity i can be inferred from $\xi_{i,\tau}$. Due to the pseudorandomness of Prf, the tag is indistinguishable from a random value for any client who does not know the identity i, thus ensuring that the tag does not reveal any information about i.

Therefore, since the commitment, ZKP, random values, and anonymous credentials reveal no information about the client's identity, the entire scheme guarantees anonymity.

□

Thus, under Assumption 1, Theorem 1 holds. □

B. Identify Malicious

Definition (three-move public-coin zero-knowledge proof) The protocol is a two-party interaction between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ which is called a three-step public-key-protocol for relation $\mathcal{R}$ with challenge set $\mathcal{C}$, instance w and witness x, if it satisfies the following properties:

- **3-move form**: 1) $\mathcal{P}$ generates and sends commitments to $\mathcal{V}$; 2) $\mathcal{V}$ selects and sends a challenge to $\mathcal{P}$; 3). $\mathcal{P}$ calculates and sends responses to $\mathcal{V}$. $\mathcal{V}$ then determines whether to accept or reject based on the protocol transcript (commitments, challenge, and responses).
- **Completeness:** If $(w, x) \in \mathcal{R}$, $\mathcal{V}$ will accept the proof.
- **Soundness:** Any PPT prover that does not know the witness should have a negligible probability of convincing an honest verifier to accept a false instance.
- **Zero-Knowledge:** There exists a PPT simulator $\mathcal{S}$, which, given a challenge c from challenge set $\mathcal{C}$, produces triples (t, d, s) whose distribution is computationally indistinguishable from the distribution of the protocol's real transcripts.

Theorem 2 (Identify Malicious Security)**.** *The protocol $\pi_{s_{ij}}$ described in Section VI is a three-move public-coin zero-knowledge proof , under the cryptographic assumptions in Assumption 1.*

Proof. 3-move form. This is trival. □

Completeness. The verifier checks the right-hand side (RHS) of the verification equation:

$$\begin{aligned}
&\mathsf{Com}(\mathbf{m}_i, \mathbf{s}_i)^c \mathsf{Com}(0, \mathbf{r}_i)^b \cdot \prod_j \mathsf{Com}(\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j)^{\delta_{ij}} \mathsf{Com}(0, \mathbf{s}_{ij})^{b\delta_{ij}} \\
&= \mathbf{g}^{c\mathbf{m}_i} \mathbf{h}^{c\mathbf{s}_i + b\mathbf{r}_i} \cdot \prod_j \left(\mathbf{g}^{\delta_{ij}\boldsymbol{\alpha}_j} \mathbf{h}^{\delta_{ij}\boldsymbol{\beta}_j} \cdot \mathbf{g}^0 \mathbf{h}^{b\delta_{ij}\mathbf{s}_{ij}} \right) \\
&= \mathbf{g}^{c\mathbf{m}_i + \sum_j \delta_{ij}\boldsymbol{\alpha}_j} \cdot \mathbf{h}^{c\mathbf{s}_i + b\mathbf{r}_i + \sum_j \delta_{ij}(\boldsymbol{\beta}_j + b\mathbf{s}_{ij})} = \mathrm{RHS}.
\end{aligned}$$

The left-hand side (LHS) of the verification equation is:

$$\begin{aligned}
\mathrm{LHS} &= \mathsf{Com}(\mathbf{R}_m, \mathbf{R}_s) = \mathbf{g}^{\mathbf{R}_m} \mathbf{h}^{\mathbf{R}_s} \\
&= \mathbf{g}^{c\mathbf{m}_i + \sum \delta_{ij}\alpha_j} \mathbf{h}^{(c+b)\mathbf{s}_i + \sum \delta_{ij}\beta_j}.
\end{aligned}$$

The exponents of $\mathbf{g}$ on both sides match. Since $\mathbf{r}_i + \sum_j \delta_{ij}\mathbf{s}_{ij} = \mathbf{s}_i$, the exponent of $\mathbf{h}$ can be rewritten as:

$$\mathrm{LHS} = (c+b)\mathbf{s}_i + \sum_j \delta_{ij}\beta_j = c\mathbf{s}_i + b\mathbf{r}_i + \sum_j \delta_{ij}(\boldsymbol{\beta}_j + b\mathbf{s}_{ij}) = \mathrm{RHS}.$$

The verifier accepts the proof if and only if both sides of the formula are equal. So it satisfies completeness. □

Soundness. Soundness assumes that the prover $\mathcal{P}$ does not know the witness $(\mathbf{s}_i, \{\mathbf{s}_{ij}\})$ and ensures that if $\mathsf{R}_{s_{ij}}$, as defined in Eq. 12, is not satisfied, the verifier $\mathcal{V}$ will reject the proof. The verifier checks:

$$\begin{aligned}
\mathsf{Com}(\mathbf{R}_m, \mathbf{R}_s) = \mathsf{Com}(\mathbf{x}_i, \mathbf{s}_i)^c \mathsf{Com}(0, \mathbf{r}_i)^b \\
\cdot \prod_j \mathsf{Com}(\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j)^{\delta_{ij}} \mathsf{Com}(0, \mathbf{s}_{ij})^{b\delta_{ij}}.
\end{aligned}$$

Simplifying the above expression: $\mathbf{h}^{b\mathbf{s}_i} = \mathbf{h}^{b\mathbf{r}_i + b\sum_j \delta_{ij}\mathbf{s}_{ij}}$.

Since b is a challenge randomly selected by $\mathcal{V}$ from the challenge set $\mathcal{C}$, the prover $\mathcal{P}$ must know $\mathbf{s}_i$, $\mathbf{r}_i$, and $\{\mathbf{s}_{ij}\}$ to satisfy this equation. However, $\mathcal{P}$ only knows $\mathbf{r}_i$ and does not know the witness $(\mathbf{s}_i, \{\mathbf{s}_{ij}\})$.

A prover without the correct witness $(\mathbf{s}_i, \{\mathbf{s}_{ij}\})$ cannot compute valid $\mathbf{R}_m$ and $\mathbf{R}_s$ to satisfy the verification equation, due to the binding property of the Pedersen commitment and the unpredictability of the challenge b chosen by $\mathcal{V}$.

Hence, a dishonest prover succeeds only with negligible probability, ensuring soundness.

□

Zero-knowledge. Assume a simulator $\mathcal{S}$ which generates a triple (t, d, s) that is indistinguishable from a real protocol $\pi_{s_{ij}}$'s transaction (commitments, challenges, responses), where commitments are $\{\mathsf{Com}(\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j)\}_j$, $\{\mathsf{Com}(0, \mathbf{s}_{ij})\}_j$, the challenges are (c, b), the responses are $(\mathbf{R}_m, \mathbf{R}_s)$, using only challenges (c, b) and the instance $(\mathsf{Com}(\mathbf{x}_i, \mathbf{s}_i), \mathbf{r}_i)$.

The triple (t, d, s) are constructed as: 1) $\mathcal{S}$ sets (c, b) as challenge d. 2) $\mathcal{S}$ randomly select $(\{\mathbf{t}_j^{(1)}, \mathbf{t}_j^{(2)}\}_j)$ as commitments t. 3) $\mathcal{S}$ randomly selects $\mathbf{s}_1$, and uses it and t, d to compute $\mathbf{s}_2 = \mathsf{Com}(\mathbf{x}_i, \mathbf{s}_i)^c (\mathbf{t}^{(0)})^b \prod_j (\mathbf{t}_j^{(1)})^{\delta_{ij}} (\mathbf{t}_j^{(2)})^{b\delta_{ij}} / \mathbf{s}_1$. Then, $\mathcal{S}$ sets $(\mathbf{s}_1, \mathbf{s}_2)$ as response s. This guarantees that the triple will pass the verification.

For the commitments, $\{\mathbf{t}_j^{(1)}\}_j$ are randomly generated, and $\{\mathsf{Com}(\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j)\}_j$ are also random. Therefore, they are indistinguishable. For $\{\mathsf{Com}(0, \mathbf{s}_{ij})\}_j$, $\mathbf{s}_{ij}$ is the witness, which is unknown to the verifier $\mathcal{V}$, and $\mathbf{s}_{ij}$ itself is generated using a random seed. Hence, just like $\mathbf{t}_j^{(2)}$, it is indistinguishable to $\mathcal{V}$. Therefore, the commitment and t are indistinguishable.

For the challenge, similarly, $(\mathbf{R}_m, \mathbf{R}_s)$ is a linear combination of $\mathbf{x}_i, \mathbf{s}_i$ from the witness and the random values $\{\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j\}_j$, along with the challenge c, b. Apart from c and b, it is unknown to $\mathcal{V}$. Thus, it is indistinguishable from $(\mathbf{s}_1, \mathbf{s}_2)$, which is also computed randomly. Therefore, the response and s are indistinguishable.

Consequently, the triple (t, d, s) is computationally indistinguishable from the real protocol $\pi_{s_{ij}}$'s transaction (commitments, challenges, responses). Thus, Protocol $\pi_{s_{ij}}$ is a three-move public-coin ZKP.

□

□

VIII. Evaluation

A. Overhead under Different Configurations

1) Forbid Malicious Client in Anonymity Tests: This experiment evaluates the time overhead of using a blacklist to prevent malicious clients in blk^q from rejoining the aggregation process under anonymity. The experiment's primary parameters are divided into two parts: the size M of the newly added q-level blacklist blk_τ^q in the current round, and the cumulative number n of q-level blacklists from the previous τ rounds.

For testing the variable M, we assume 14 q-th blacklists have already been added in prior rounds, resulting in a total of 16 proofs to be aggregated. These include 14 previously proven blacklists $\mathsf{blk}_z^q \in \mathsf{blk}^q$, one joint proof $R_{\text{anc}}, R_{\text{isu}}$ (for acceleration), and one new proof for the blacklist blk_τ^q. The experimental results, shown in Figure 6-(a), indicate that only the proof R_{blk} has a linear relationship with the blacklist length $|\mathsf{blk}_\tau^q| = M$, while the time overhead of other proofs or verifications remains unaffected by M, as their inputs are independent of blk_τ^q. Since these proofs can be transformed into non-interactive proofs using the Fiat-Shamir [57] transformation, only one transmission is needed. As shown in Figure 6-(c), bandwidth remains independent of M because the only item sent is π_{zkbl}, whose size does not depend on M. For testing the variable $n = |\mathsf{blk}|$, since the generation of $R_{\text{isu}}, R_{\text{anc}}, R_{\text{blk}}$ is independent of n, we only tested the generation and verification of R_{zkbl}. The results, shown in Figure 6-(b), reveal that R_{zkbl} includes the aggregated proof of n blacklists, as well as the proofs R_{isu} and R_{anc}, totaling $n + 2$ proofs, which creates a linear relationship. Similarly, the verification of R_{zkbl} also shows a linear relationship, as illustrated by VERIFY in Figure 6.

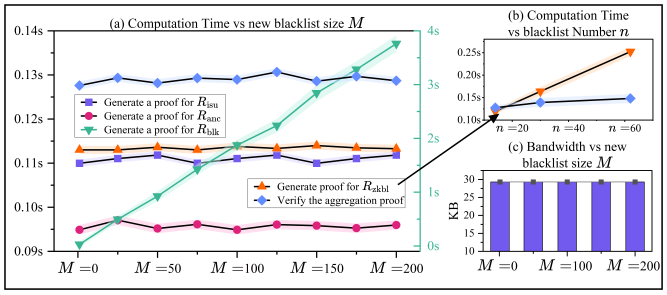

Fig. 6: Time Computation and Bandwidth for Blacklist in WhiteCloak with blacklist size M and aggregated proofs number n. For (a) and (c), $n = 16$, for (b), $M = 100$.

2) Remove Malicious Client in Anonymity Tests: The experiment consists of two parts. In the first part, "Removal of Malicious Client inputs by clients and server," we measured the time overhead concerning the following four variables: data dimension $\ell = \{10^3, 10^4, 10^5\}$, the number of malicious clients in the blacklist $M = \{1, 10, 100\}$, the threshold $t = \{10, 50, 100\}$ for secret sharing used in SA, and the total number of participating clients $N = \{10^3, 10^4, 10^5\}$. The time measured includes the server's runtime and the total time for all participating clients (client time is serialized, not parallelized). The experimental results are shown in Figure 7 (a)-(d). The time overhead has minimal dependency on data dimension ℓ, as it involves only addition and subtraction of ℓ-dimensional vectors, making little impact on the result. The time overhead is linearly proportional to the number of malicious clients M, as it relates to the number of inputs and key shares to be removed. The overhead scales exponentially with the secret sharing threshold t, due to the $O(t^2)$ complexity of Shamir secret recovery operations. Finally, the time overhead scales logarithmically with the total number of clients N, as each client's number of neighbors is $O(\log N)$. These results

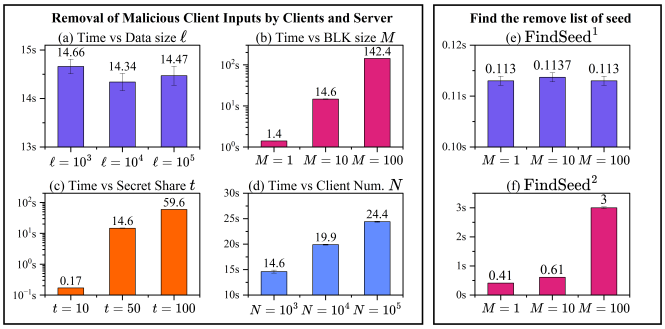

Fig. 7: Performance Evaluation of WhiteCloak's Malicious Client Input Removal Across Different Seed Retrieval Scenarios and Parameter Settings. The default parameters are $\ell = 10^3, M = 10, t = 50, N = 10^3$.

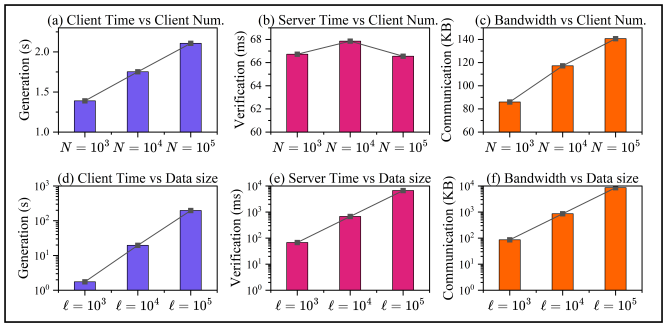

Fig. 8: Performance of Protocol $\pi_{s_{ij}}$ in Identifying Malicious Clients through SA Key Validation. The default parameters are $\ell = 10^3, N = 10^3$.

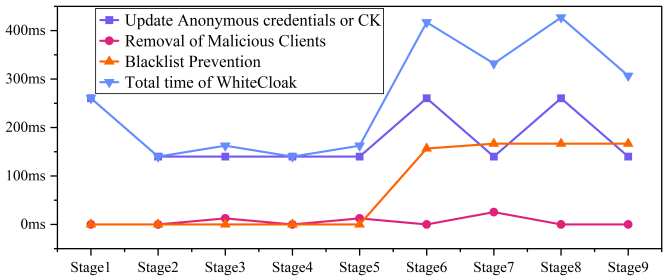

Fig. 9: WhiteCloak introduces anonymity to secure aggregation, utilizing blacklisting to block malicious clients and remove their inputs, resulting in additional system overhead.

are consistent with theoretical predictions.

For the second part "Find the remove list of seed" operation in the three scenarios, in the one interaction round case, the anonymous credential of the seed matches that of the participating client, so no additional seed search is needed. The experimental results are shown in Figure 7 (e)-(f). For FindSeed^1, a Groth16 proof is generated for each instance, verifying that the anonymous credential of the seed matches that of the client for the same ID i, and thus is independent of the number of malicious clients M. In the case of FindSeed^2, a series of proofs must be generated to ensure that the new anonymous credential is valid and not in the blacklist blk, where $|\mathsf{blk}| = M$. As M increases, the input size for the corresponding Groth16 proofs also grows. According to the experimental results, in the "one seed in multiple interaction rounds" scenario, FindSeed^2 is preferable when there are fewer malicious clients or when renegotiating the random seed is costly. When the number of malicious clients is high, executing FindSeed^1 is more appropriate.

Since the time overhead for determining which p-th blacklist a malicious client should be added to is similar to the time shown in Figure 6 (a), an additional plot is not provided here. The only difference is that this process requires generating proofs solely for R_{blk} and R_{zkbl}, without the need for other proofs. Furthermore, for previously generated blacklists, the corresponding R_{blk} proof does not need to be regenerated. Thus, the time for this function can be directly referenced from Figure 6.

3) Detect Malicious Clients' key Tests: In this experiment, we tested the protocol $\pi_{s_{ij}}$ proposed in Section VI, which serves as a supplement to RoFL [16] and ACORN [17] within WhiteCloak, aiming to identify the specific client using an incorrect SA key. The protocol's performance is primarily influenced by two variables: the number of clients participating in the aggregation N (where each client's key is composed of key shares from $\log N$ clients) and the data dimension ℓ (where the key dimension matches the data dimension). Accordingly, we set $N = \{10^3, 10^4, 10^5\}$ and $\ell = \{10^3, 10^4, 10^5\}$. The experimental results are shown in Figure 8. The time taken for a client to generate a proof has a logarithmic relationship with the total number of clients N, at $O(\log N)$, while the server's verification time and bandwidth are independent of the number of clients, as the responses $\mathbf{R}_m, \mathbf{R}_s$ sent to the server by the client do not depend on N. The experiment validates the theoretical correctness of this relationship. The time for a client to generate a proof is linearly proportional to the data dimension ℓ, as the time required to generate a constant Pedersen commitment is $O(\log q)$, so for ℓ dimensions, it is $O(\ell \log q)$. Similarly, the server's verification time and bandwidth exhibit a linear relationship with ℓ, as the responses $\mathbf{R}_m, \mathbf{R}_s$ sent by the client to the server are ℓ-dimensional.

4) The impact when malicious clients are added to the blacklist: We set the maximum blacklist level to $q = 2$, meaning that clients reaching the second-level blacklist are permanently prohibited from participating in future aggregation processes. The total number of participating clients is $N = 10^2$, the dimension of the data sent is $\ell = 10^5$, and the threshold for secret sharing is $t = 10$. SA based on Bell's [31] one seed within one interaction round was used. The experiment is designed to examine the cumulative impact of different blacklist configurations and malicious client behaviors on system resource consumption throughout the continuous process:

(1) Clients register with the issuer. (2) $|\mathsf{blk}^q| = 0, M = 0$: Clients update their anonymous credentials. (3) $|\mathsf{blk}^q| = 0$, $M = 5$: Five malicious clients are detected and added to

blk^1, excluding their inputs from the aggregation. Clients update their anonymous credentials. (4) $|\mathsf{blk}^q| = 0, M = 0$: No malicious clients are present, and clients update their anonymous credentials. (5) $|\mathsf{blk}^q| = 0, M = 5$: 5 malicious clients are detected, one of whom is a repeat offender. The repeat offender is moved to blk^q, while the remaining four are added to blk^1. All malicious inputs are excluded, and clients update their anonymous credentials. (6) $|\mathsf{blk}^q| = 1$, $M = 0$: Updates parameters for the new blk^q, and clients update their anonymous credentials. (7) $|\mathsf{blk}^q| = 1$, $M = 9$: 9 malicious clients, all repeat offenders, are detected and added to blk^q, excluding their inputs from the aggregation. Clients update their anonymous credentials. (8) $|\mathsf{blk}^q| = 10$, $M = 0$: Updates parameters to for the new blk^q. Clients update their anonymous credentials. (9) $|\mathsf{blk}^q| = 10$, $M = 0$: Clients update their anonymous credentials.

The experimental results are presented in Figure 9. Each time blk^q is updated, the total number of proofs n to be aggregated also increases, necessitating the regeneration of HICIAP public parameters ck and srs. When a client generates a proof for blk^q in Stage 6, this proof can be reused in subsequent stages. However, it is still necessary to verify that the ID i used in the proof matches the one in the newly generated anonymous credential, resulting in a fixed time overhead for each round.

B. Overhead Comparison with state-of-the-art methods

In our testing framework, we adopted the same experimental setup as ACORN [17], with $N = 500$, 50 dropout clients, and 50 malicious clients. For WhiteCloak, since malicious clients are prohibited from participating in aggregation, we assume $|\mathsf{blk}| = 25$, $|\mathsf{blk}^q| = 25$. WhiteCloak uses the same input validation as ACORN and RoFL, so performance under attack is identical. The difference is that WhiteCloak works under anonymity and still prevents re-entry by malicious clients, which ACORN or RoFL [16] cannot do if made anonymous. To evaluate federated learning (FL) across different data modalities, we conducted experiments on four datasets: 1) MNIST [58] using CNN [59], 2) CIFAR10-S [60] using LeNet-5 [61], 3) CIFAR10-L [60] using ResNet [62], and 4) Shakespeare using LSTM [63]. Client time is based on the maximum client time, representing the duration required by the slowest client to recover keys for all malicious clients.

The experimental results are presented in Table II. WhiteCloak incorporates the FL model training process, SA key generation and secret recovery, and the input validation process from ACORN, while adding a blacklist mechanism to block malicious anonymous clients and remove their inputs. The results show that WhiteCloak's additional overhead is comparable to the FL training time on small datasets, such as MNIST, and is only 3% of the time required for RoFL and 21% for ACORN. This gap widens as the dataset size increases, with WhiteCloak's added anonymity mechanism scaling minimally with data dimensions. For example, on the Shakespeare dataset, WhiteCloak's additional overhead is only 1.77s, compared to 284.16s for training and 235.28s for verification, making the overhead nearly negligible. It is just 0.07% of the time required for RoFL and 0.34% for ACORN. Therefore, WhiteCloak is highly suitable for the FL scenarios in common SA. Other SA scenarios with similar data dimensions can also benefit from it.

TABLE II: $N = 500$, $|\mathsf{blk}| = 25$, $|\mathsf{blk}^q| = 25$, $t = 50$. The result for RoFL and ACORN are derived from [16], [17].

Method	Max Client time/bandwidth for one round			
	MNIST	CIFAR-10 S	CIFAR-10 L	SHAKESPEARKE
RoFL	47s	110s	800s	2526s
	4.5MB	14.2MB	62.4MB	186.6MB
ACORN	6.65s	18.81s	100.33s	519.92s
	5.68MB	11.20MB	77.97MB	29.12MB
WhiteCloak	8.07s	20.23s	101.86s	521.69s
	5.71MB	11.23MB	78MB	29.15MB

Overhead introduced by WhiteCloak is Blacklist and Input Removal.
The cost of each part of the WhiteCloak system for one round

FL Train	Blacklist	SA	Verify ℓ_2	Input Removal
		MNIST (19kparams,160rounds)		
1.17s	0.76s	0.02s	5.48s	0.66s
None	34.9KB	5.41MB	0.27MB	0.78KB
		CIFAR-10S (62kparams,100rounds)		
0.83s	0.76s	0.03s	17.95s	0.66s
None	34.9KB	11.02MB	0.18MB	0.78KB
		CIFAR-10L (273kparams,160rounds)		
20.75s	0.76s	0.16s	79.42s	0.77s
None	34.9KB	77.66MB	0.31MB	0.78KB
		Shakespeare (818kparams,20rounds)		
284.16s	0.76s	0.48s	235.28s	1.01s
None	34.9KB	29.08MB	0.15MB	0.78KB

IX. Conclusions

WhiteCloak is the first secure aggregation scheme supporting cross-round accountability under anonymity. Each anonymous client must provide a zero-knowledge proof showing they are not banned. To prevent false positives and offer clients a degree of tolerance, we allowing clients to make up to q mistakes before a ban is enforced. These features collectively protect both input and identity privacy, without linking the ban status to identifiable client information or associating anonymous credentials across aggregation rounds. Furthermore, WhiteCloak introduces small additional overhead—only 1.42s and 0.03KB in an FL scenario with 500 clients—remaining independent of model dimension and well-suited for high-dimensional data scenarios. Future work could explore extending to a fully malicious server threat model, developing decentralized credential issuance schemes to eliminate single-point trust, and optimizing the blacklist mechanism to reduce overhead growth.

X. Ethical Considerations

We believe that our work on WhiteCloak does not pose any significant ethical risks. The primary goal of WhiteCloak is to improve the security and privacy of federated learning by protecting against malicious clients while maintaining

user anonymity. The system ensures that malicious users can be detected and blacklisted without revealing any sensitive information about honest users. The research does not involve human subjects or the collection of personal data, and it does not introduce any risks to individuals or organizations. Furthermore, WhiteCloak operates within the constraints of existing privacy and security mechanisms, adhering to widely accepted cryptographic protocols, such as zero-knowledge proofs and secure aggregation techniques. We have taken care to design the system in a way that prevents abuse or misuse while ensuring accountability for malicious behavior. No ethical or legal issues related to the implementation, use, or publication of this research are foreseen.

ACKNOWLEDGMENT

This work is supported by the Major Research Plan of Hubei Province under Grant/Award NO. 2023BAA027, Changsha Key Science and Technology Project under the "Unveiling and Commanding" Initiative under Grant No. kq2503009, the project of Science, Technology and Innovation Commission of Shenzhen Municipality of China under Grant No. GJHZ20240218114659027 and the Key Research & Development Plan of Hubei Province of China under Grant No. 2024BAB049.

REFERENCES

[1] K. Talwar, S. Wang, A. McMillan, V. Jina, V. Feldman, B. Basile, Á. Cahill, Y. S. Chan, M. Chatzidakis, J. Chen, O. Chick, M. Chitnis, S. Ganta, Y. Goren, F. Granqvist, K. Guo, F. Jacobs, O. Javidbakht, A. Liu, R. Low, D. Mascenik, S. Myers, D. Park, W. Park, G. Parsa, T. Pauly, C. Priebe, R. Rishi, G. N. Rothblum, M. Scaria, L. Song, C. Song, K. Tarbe, S. Vogt, L. Winstrom, and S. Zhou, "Samplable anonymous aggregation for private federated data analysis," CoRR, vol. abs/2307.15017, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.15017

[2] D. Archer et al., "Applications of homomorphic encryption," HomomorphicEncryption.org, Redmond WA, Tech. Rep., 2017.

[3] F. Inc., "Role of applied cryptography in a privacy-focused advertising ecosystem request for proposals," https://research.fb.com/programs/research-awards/proposals/cryptography-rfp-2019/, 2020, facebook.

[4] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, 2017.

[5] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019. IEEE, 2019, pp. 691–706. [Online]. Available: https://doi.org/10.1109/SP.2019.00029

[6] J. Ye, A. Maddi, S. K. Murakonda, V. Bindschaedler, and R. Shokri, "Enhanced membership inference attacks against machine learning models," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 3093–3106. [Online]. Available: https://doi.org/10.1145/3548606.3560675

[7] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 603–618. [Online]. Available: https://doi.org/10.1145/3133956.3134012

[8] J. Takeshita, R. Karl, T. Gong, and T. Jung, "SLAP: simpler, improved private stream aggregation from ring learning with errors," J. Cryptol., vol. 36, no. 2, p. 8, 2023. [Online]. Available: https://doi.org/10.1007/s00145-023-09450-w

[9] D. Pasquini, D. Francati, and G. Ateniese, "Eluding secure aggregation in federated learning via model inconsistency," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 2429–2443. [Online]. Available: https://doi.org/10.1145/3548606.3560557

[10] E. Shi, T. H. Chan, E. G. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011. The Internet Society, 2011. [Online]. Available: https://www.ndss-symposium.org/ndss2011/privacy-preserving-aggregation-of-time-series-data

[11] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "Elsa: Secure aggregation for federated learning with malicious actors," in 2023 IEEE Symposium on Security and Privacy (SP), 2023, pp. 1961–1979.

[12] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via boolean shares," in Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings, ser. Lecture Notes in Computer Science, C. Galdi and S. Jarecki, Eds., vol. 13409. Springer, 2022, pp. 516–539. [Online]. Available: https://doi.org/10.1007/978-3-031-14791-3_23

[13] Z. Lu, S. Lu, X. Tang, and J. Wu, "Robust and verifiable privacy federated learning," IEEE Transactions on Artificial Intelligence, pp. 1–14, 2023.

[14] Z. Lu, S. Lu, Y. Cui, X. Tang, and J. Wu, "Split aggregation: Lightweight privacy-preserving federated learning resistant to byzantine attacks," IEEE Transactions on Information Forensics and Security, vol. 19, pp. 5575–5590, 2024.

[15] Z. Lu, S. Lu, Y. Cui, J. Wu, H. Nie, J. Xiao, and Z. Yi, "Lightweight byzantine-robust and privacy-preserving federated learning," in Euro-Par 2024: Parallel Processing, J. Carretero, S. Shende, J. Garcia-Blas, I. Brandic, K. Olcoz, and M. Schreiber, Eds. Cham: Springer Nature Switzerland, 2024, pp. 274–287.

[16] 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 (SP), 2023, pp. 453–476.

[17] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun, "ACORN: Input validation for secure aggregation," in 32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association, Aug. 2023, pp. 4805–4822. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/bell

[18] Y. Zhu, Y. Wu, Z. Luo, B. C. Ooi, and X. Xiao, "Secure and verifiable data collaboration with low-cost zero-knowledge proofs," Proc. VLDB Endow., vol. 17, no. 9, p. 2321–2334, May 2024. [Online]. Available: https://doi.org/10.14778/3665844.3665860

[19] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. IEEE, 2022, pp. 1354–1371. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833647

[20] V. Shejwalkar and A. Houmansadr, "Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning," in 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021. The Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/manipulating-the-byzantine-optimizing-model-poisoning-attacks-and-defenses-for-federated-learning/

[21] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 1605–1622. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/fang

[22] L. Rainie, Sara, Kiesler, R. Kang, and M. Madden, "Anonymity, privacy, and security online," 2013. [Online]. Available: https://www.pewresearch.org/internet/2013/09/05/anonymity-privacy-and-security-online/

[23] P. R. Center, "How americans view data privacy: Tech companies, ai, regulation, passwords and policies," 2023. [Online]. Available: https://www.pewresearch.org/internet/2023/05/09/how-americans-view-data-privacy/

[24] C. McClain, M. Faverio, M. Anderson, and E. Park, "How americans view data privacy," 2023. [Online]. Available: https://www.pewresearch.org/internet/2023/10/18/how-americans-view-data-privacy/

[25] S. A. Kakvi, K. M. Martin, C. Putman, and E. A. Quaglia, "Sok: Anonymous credentials," in Security Standardisation Research - 8th International Conference, SSR 2023, Lyon, France, April 22-23, 2023, Proceedings, ser. Lecture Notes in Computer Science, F. Günther and J. Hesse, Eds., vol. 13895. Springer, 2023, pp. 129–151. [Online]. Available: https://doi.org/10.1007/978-3-031-30731-7_6

[26] X. Yuan, J. Liu, B. Wang, W. Wang, B. Wang, T. Li, X. Ma, and W. Pedrycz, "Fedcomm: A privacy-enhanced and efficient authentication protocol for federated learning in vehicular ad-hoc networks," IEEE Transactions on Information Forensics and Security, vol. 19, pp. 777–792, 2024.

[27] Y. Zhang, Q. Chen, and S. Zhong, "Privacy-preserving data aggregation in mobile phone sensing," IEEE Transactions on Information Forensics and Security, vol. 11, no. 5, pp. 980–992, 2016.

[28] M. Kohlweiss, A. Lysyanskaya, and A. Nguyen, "Privacy-preserving blueprints," in Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II, ser. Lecture Notes in Computer Science, C. Hazay and M. Stam, Eds., vol. 14005. Springer, 2023, pp. 594–625. [Online]. Available: https://doi.org/10.1007/978-3-031-30617-4_20

[29] J. Groth, "On the size of pairing-based non-interactive arguments," in Advances in Cryptology – EUROCRYPT 2016, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326.

[30] M. Rosenberg, M. Maller, and I. Miers, "Snarkblock: Federated anonymous blocklisting from hidden common input aggregate proofs," in 2022 IEEE Symposium on Security and Privacy (SP), 2022, pp. 948–965.

[31] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 1253–1269. [Online]. Available: https://doi.org/10.1145/3372297.3417885

[32] K. A. 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 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1175–1191. [Online]. Available: https://doi.org/10.1145/3133956.3133982

[33] 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 (SP), 2023, pp. 477–496.

[34] H. Karthikeyan and A. Polychroniadou, "OPA: One-shot private aggregation with single client interaction and its applications to federated learning," Cryptology ePrint Archive, Paper 2024/723, 2024. [Online]. Available: https://eprint.iacr.org/2024/723

[35] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten, "Eiffel: Ensuring integrity for federated learning," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 2535–2549. [Online]. Available: https://doi.org/10.1145/3548606.3560611

[36] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017, A. Akella and J. Howell, Eds. USENIX Association, 2017, pp. 259–282. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs

[37] C. Gentry, S. Halevi, and V. Lyubashevsky, "Practical non-interactive publicly verifiable secret sharing with thousands of parties," in Advances in Cryptology – EUROCRYPT 2022, O. Dunkelman and S. Dziembowski, Eds. Cham: Springer International Publishing, 2022, pp. 458–487.

[38] Z. Lu and S. Lu, "LZKSA: Lattice-based special zero-knowledge proofs for secure aggregation's input verification," Cryptology ePrint Archive, Paper 2025/1141, 2025. [Online]. Available: https://eprint.iacr.org/2025/1141

[39] J. Brorsson, B. David, L. Gentile, E. Pagnin, and P. S. Wagner, "PAPR: publicly auditable privacy revocation for anonymous credentials," in Topics in Cryptology - CT-RSA 2023 - Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24-27, 2023, Proceedings, ser. Lecture Notes in Computer Science, M. Rosulek, Ed., vol. 13871. Springer, 2023, pp. 163–190. [Online]. Available: https://doi.org/10.1007/978-3-031-30872-7_7

[40] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "Blac: Revoking repeatedly misbehaving anonymous users without relying on ttps," ACM Trans. Inf. Syst. Secur., vol. 13, no. 4, Dec. 2010. [Online]. Available: https://doi.org/10.1145/1880022.1880033

[41] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in SIGMOD '10: Proceedings of the 2010 ACM SIGMOD international conference on Management of data. Association for Computing Machinery, Inc., January 2010. [Online]. Available: https://www.microsoft.com/en-us/research/publication/differentially-private-aggregation-of-distributed-time-series-with-transformation-and-encryption-2/

[42] G. Ács and C. Castelluccia, "I have a dream! (differentially private smart metering)," in Information Hiding - 13th International Conference, IH 2011, Prague, Czech Republic, May 18-20, 2011, Revised Selected Papers, ser. Lecture Notes in Computer Science, T. Filler, T. Pevný, S. Craver, and A. D. Ker, Eds., vol. 6958. Springer, 2011, pp. 118–132. [Online]. Available: https://doi.org/10.1007/978-3-642-24178-9_9

[43] 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, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. IEEE Computer Society, 2018, pp. 315–334. [Online]. Available: https://doi.org/10.1109/SP.2018.00020

[44] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in 28th Annual Symposium on Foundations of Computer Science (sfcs 1987), 1987, pp. 427–438.

[45] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, O. Goldreich, Ed. ACM, 2019, pp. 329–349. [Online]. Available: https://doi.org/10.1145/3335741.3335757

[46] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," CoRR, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[47] B. Defend and K. Kursawe, "Implementation of privacy-friendly aggregation for the smart grid," in SEGS'13, Proceedings of the 2013 ACM Workshop on Smart Energy Grid Security, Co-located with CCS 2013, November 8, 2013, Berlin, Germany, B. Defend and K. Kursawe, Eds. ACM, 2013, pp. 65–74. [Online]. Available: https://doi.org/10.1145/2516930.2516936

[48] A. Shamir, "How to share a secret," Commun. ACM, vol. 22, no. 11, pp. 612–613, 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[49] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed., vol. 576. Springer, 1991, pp. 129–140. [Online]. Available: https://doi.org/10.1007/3-540-46766-1_9

[50] C. Schnorr, "Efficient identification and signatures for smart cards," in Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer, 1989, pp. 239–252. [Online]. Available: https://doi.org/10.1007/0-387-34805-0_22

[51] C. P. Schnorr, "Efficient signature generation by smart cards," J. Cryptol., vol. 4, no. 3, pp. 161–174, 1991. [Online]. Available: https://doi.org/10.1007/BF00196725

[52] D. Chaum, "Untraceable electronic mail, return addresses, and digital

pseudonyms," Commun. ACM, vol. 24, no. 2, pp. 84–88, 1981. [Online]. Available: https://doi.org/10.1145/358549.358563
[53] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA, M. Blaze, Ed. USENIX, 2004, pp. 303–320.
[54] S. A. Kakvi, K. M. Martin, C. Putman, and E. A. Quaglia, "Sok: Anonymous credentials," in Security Standardisation Research, F. Günther and J. Hesse, Eds. Cham: Springer Nature Switzerland, 2023, pp. 129–151.
[55] D. Boneh and X. Boyen, "Short signatures without random oracles," in Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer, 2004, pp. 56–73. [Online]. Available: https://doi.org/10.1007/978-3-540-24676-3_4
[56] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Technical Report/ETH Zurich, Department of Computer Science, vol. 260, 1997.
[57] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in Advances in Cryptology — CRYPTO' 86, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.
[58] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.
[59] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," CoRR, vol. abs/1812.01097, 2018. [Online]. Available: http://arxiv.org/abs/1812.01097
[60] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, 05 2012.
[61] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://doi.org/10.1109/5.726791
[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90
[63] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735
[64] D. Cryptography, "Rust curve25519 library," August 2020, [online]. [Online]. Available: https://github.com/dalek-cryptography/curve25519-dalek
[65] ChrisMacNaughton and L. Bruno, "Pure rust implementation of shamir's secret sharing," https://github.com/Nebulosus/shamir, 2023, accessed: 2024-11-05.
[66] C. Y. Henry de Valence and O. Andreev, "A pure-rust implementation of bulletproofs using ristretto." https://github.com/zkcrypto/bulletproofs, 2023, accessed: 2024-11-05.
[67] arkworks rs, "arkworks-rs/groth16," 2024, accessed: 2024-11-09. [Online]. Available: https://github.com/arkworks-rs/groth16
[68] M. Rosenberg, "rozbb/hiciap," 2024, accessed: 2024-11-09. [Online]. Available: https://github.com/rozbb/hiciap
[69] L. B. Hidde Lycklama, "pps-lab/fl-analysis," 2024, accessed: 2024-11-09. [Online]. Available: https://github.com/pps-lab/fl-analysis
[70] M. Rosenberg, M. Maller, and I. Miers, "SNARKBlock: Federated anonymous blocklisting from hidden common input aggregate proofs," Cryptology ePrint Archive, Paper 2021/1577, 2021. [Online]. Available: https://eprint.iacr.org/2021/1577

APPENDIX

A. Experimental Environment

Our experimental platform is a desktop computer equipped with an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz and 32GB of RAM. The WhiteCloak implementation is built using RUST 1.84, with the encryption module based on Curve25519 [64], secret sharing implemented as described in [65], Pedersen commitments based on [66], and Groth16 following the implementation in [67]. The HICIAP component uses Rosenberg's implementation [68]. Each data dimension is represented with 32-bit precision. The federated learning implementation is based on Python, following the approach of Lycklama et al. [69]. The additional performance overhead introduced by WhiteCloak is the average value obtained from 10 experiments. For the neural networks of FL, the configurations are as follows:

- CNN [59] with parameters $\ell = 19k, \tau_{\max} = 160$.
- LeNet-5 [61] with parameters $\ell = 62k, \tau_{\max} = 100$.
- ResNet [62] with parameters $\ell = 273k, \tau_{\max} = 160$.
- LSTM [63] with parameters $\ell = 818k, \tau_{\max} = 20$.

B. Instantiation for Detecting Malicious Clients Using New Credentials

To better understand the core concept of the WhiteCloak protocol, which ensures that "even if the client updates the anonymous credentials in each round, the server can still detect the client that was previously added to the blacklist without knowing the client's identity," we instantiate the protocol's operations here. This allows for an intuitive understanding of how WhiteCloak updates credentials, generates proofs, links proofs, and verifies these proofs.

Assume that there are two clients in the system, with IDs 1, 2, and the system is in round τ, set blacklist level $q = 1$. At this point, the blacklist contains only the anonymous credential and random number pair $\{\mathsf{Prf}_2(\xi_{2,\nu}), \xi_{2,\nu}\}$ generated by client 2 in round ν. The WhiteCloak protocol proceeds as follows:

Client i first updates its anonymous credential to $\mathsf{Prf}_i(\xi_{i,\tau})$, and uses the Groth16 proving system to generate three proofs $\pi_{\text{isu}}, \pi_{\text{anc}}, \pi_{\text{blk}}$. The detailed format for each proof can be found in Section 2.1. Notably, for the proof π_{blk}, it serves to verify whether $\mathsf{Prf}_2(\xi_{2,\nu})$ was generated by client i. Thus, for client 2, this verification will fail; however, for client 1, this verification will succeed. For the proofs π_{isu} and π_{anc}, their purpose is to verify whether $\mathsf{Prf}_i(\xi_{i,\tau})$ was generated using client i's registered ID. Therefore, as long as the client generates the anonymous credential using the ID registered with the Issuer, both verifications will succeed. This step effectively identifies malicious clients' new anonymousF credentials.

Since the identity information i is leaked in the above proofs, it is necessary to link the three proofs together while hiding the identity i. To achieve this, the three proofs are transformed as $\hat{\pi}_{\text{isu}}, \hat{\pi}_{\text{anc}}, \hat{\pi}_{\text{blk}}$. These proofs are generated through interaction with the verifier, and during this process, intermediate values are produced. These intermediate values will be used in subsequent linking proofs. For example, a commitment is generated as $\mathsf{com}_{a_0}^{(o)} \leftarrow a_0 P_1 + z_1^{(o)} P_2 + z_3^{(o)} P_3$, where P_1, P_2, P_3 are the Pedersen bases, and $(z_1^{(o)}, z_3^{(o)})$ are the opening values of $\mathsf{com}_{a_0}^{(o)}$, with a_0 being the ID i and $o \in \{\text{isu}, \text{anc}, \text{blk}\}$. The HICIAP.Link protocol, as a Σ-protocol [56], uses $\{P_i\}_{i=1}^3$, $\hat{\pi}_o$ as the instance, and $a_0, (z_1^{(o)}, z_3^{(o)})$ as the witness. It then generates random numbers $\alpha, \{\beta_o, \gamma_o\}$ and produces a new commitment $\mathsf{com}_o \leftarrow \alpha P_1 + \beta_o P_2 + \gamma_o P_3$. This commitment is sent to the verifier, who then returns a

challenge c and computes $r \leftarrow \alpha - ca_0$, $s_o \leftarrow \beta_o - cz_1^{(o)}$, $\mu_o \leftarrow \gamma_o - cz_3^{(o)}$. The verifier checks if $\text{com}_o \stackrel{?}{=} rP_1 + s_oP_2 + \mu_oP_3 + c\text{com}_{a_0}^{(o)}$. The detailed format for each proof can be found in Appendix C.

C. HICIAP details

For more detail about HICIAP.Prove, please check the Figure 4 of [30] at page 9. For more detail about HICIAP.Link, please check the Figure 9 of [70] at page 28.

- $\text{HICIAP.Prove}((\text{ck}, \text{crs}); \hat{\mathbf{S}}; (a_0, A', B', C')) \rightarrow (\hat{\pi}, o)$: Generates proof $\hat{\pi}$ and output $o = (z_1, z_3)$ where $(A', B', C') \leftarrow \text{Groth16.ReRand}^{\mathbb{M}}(A', B', C')$, and the following steps are performed: $A = A' \parallel [z_1]_1 \parallel [z_2]_1$,

$$B = B' \parallel [\gamma]_2 \parallel [\delta]_2 \in \mathbb{G}_2^n, C = C' \parallel [1]_1 \parallel [z_2]_1 \in \mathbb{G}_1^n,$$
$$\text{com}_{a_0} = a_0P_1 + z_1P_2 + z_3P_3, \text{com}_A = A * \text{ck}_1,$$
$$\text{com}_B = \text{ck}_2 * B, \text{com}_C = e([z_4]_1, \text{ck}_3) \cdot (C * \text{ck}_1),$$

  where $r = (r, r^2, \ldots, r^n)$, $r' = r_{[n-2]}$, $\text{agg}_{\text{in}} = \hat{S}^r$, $\text{agg}_C = C^r$, $W = [z_1 r^{n-1}] + \sum_{i=1}^{n-2} r_i a_0 W_0$, $G_1 = \sum_{i=1}^{n-2} r_i W_0$, $G_2 = [r^{n-1}]_1$, and $\text{agg}_{AB} = A' * B$ are the variables that interact with the verifier intermediate.
- $\text{HICIAP.Vfy}(\text{srs}, \text{crs}; \text{com}_{\text{in}}) \rightarrow \{0, 1\}$: Verifies the input commitment com_{in} and outputs 1 if valid, 0 otherwise. The verification steps are as follows:

$$J = e(\text{agg}_{\text{in}}, [\gamma]_2), G_1 = \sum_{i=1}^{n-2} r^i W_0, \quad G_2 = [r^{n-1}]_1$$
$$\text{agg}_{AB} = \prod_{i=1}^{n} e([\alpha]_1, [\beta]_2)^{r^i} \cdot J \cdot e(W, [\gamma]_2) \cdot e(\text{agg}_C, [\delta]_2)$$

- $\text{HICIAP.LinkProve}\left(\{P_i\}_{i=1}^3, \{\text{com}_a^{(i)}\}_{i=1}^t; a_0, \{z_1^{(i)}, z_3^{(i)}\}_{i=1}^t\right) \rightarrow \{\text{com}_i\}_{i=1}^t$: Produces commitments com_i for $i = 1, \ldots, t$ by performing the following steps:

$$r = \alpha - ca_0, \alpha, \beta_i, \gamma_i \leftarrow \mathbb{F}, \text{com}_i \quad = \alpha P_1 + \beta_i P_2 + \gamma_i P_3$$
$$s_i = \beta_i - cz_1^{(i)}, u_i := \gamma_i - cz_3^{(i)}$$

- $\text{HICIAP.LinkVfy}\left(\{P_i\}_{i=1}^3, \{\text{com}_a^{(i)}\}_{i=1}^t\right) \rightarrow \{0, 1\}$: Verifies that the commitments $\{\text{com}_i\}_{i=1}^t$ are valid by checking the following conditions:

$$\text{com}_i = rP_1 + s_iP_2 + u_iP_3 + c \cdot \text{com}_a^{(i)}$$

D. Tolerance for Malicious Clients

The multi-level blacklist mechanism inherently mitigates the risk of misclassification by requiring a client to accumulate at least q misbehaviors before being permanently banned. This approach reduces the likelihood of erroneous blacklisting, as it provides an opportunity for clients without facing immediate and severe consequences.

In addition to the appeal process, timed bans and automatic unbanning offer another layer of flexibility and fairness. When a client is added to the q-blacklist, a timestamp is recorded, marking the start of the ban period. Once the predefined time duration, T, has elapsed, the client will automatically be removed from the blacklist, provided they have not accumulated additional infractions during this period.

E. Misjudgment and Malicious Server

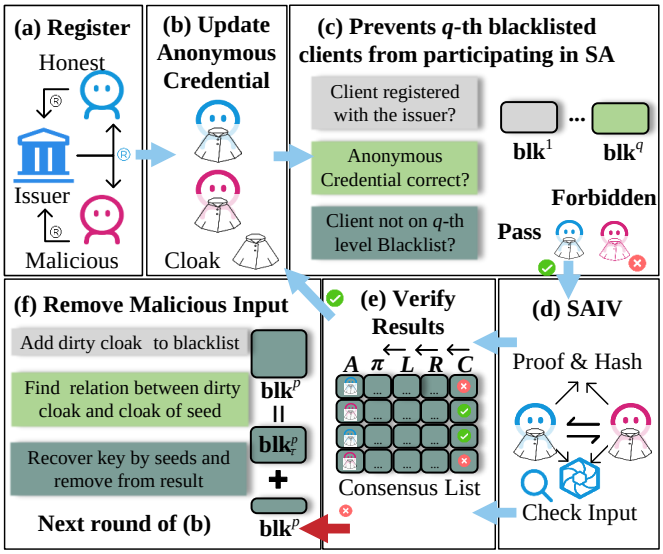

Fig. 10: (a)-(d), (e) same as Figure 1. (e) Client sends the proof π to the consensus list, while the server send left-hand side L and right-hand side R of the verification formula, sending the verification result C to the consensus list. Verify whether the linear combination of elements in π matches and L, R. If matched, the verification is correct, and the verification result is malicious, the client is marked as malicious.

To address rare misclassification events, a benign client can initiate an appeal process. The client submits the original ZKP they provided during their initial participation, along with the corresponding hash that was previously submitted to ensure data integrity and transmission authenticity. The server can then efficiently re-verify the proof against the client's behavior and history. If the proof is validated and it is determined that the client was wrongly flagged, the server can promptly remove the client from the blacklist, restoring their access to the system. This process is conceptually similar to mitigating attacks by malicious servers, where the server may misclassify honest clients as malicious or vice versa. The design of this appeal process is illustrated in Figure 10.

F. The whole WhiteCloak

Notice: Blue represent newly added modules by WhiteCloak. The red text indicates modifications we have made to the ACORN in WhiteCloak. The black text denotes original components of ACORN.

System Initialization

1) Set $\tau = 0$ and initialize $\{\mathbf{blk}_\tau^z, \mathbf{blk}^z = \emptyset\}_{z=1}^q$. Generate system parameters as follows: $\mathsf{crs}_{\mathsf{isu/anc/blk}} \leftarrow \mathsf{Groth16}.\mathsf{Setup}(R_{\mathsf{isu/anc/blk}})$ and $(\mathsf{ck}, \mathsf{srs}) \leftarrow \mathsf{HICIAP.GenCk}(n)$, where n is the number of aggregated proofs.
2) Each client registers with the issuer to generate the key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ using the Schnorr protocol. The client then commits to their identity i with a random value r_i by computing $\mathsf{com}_{i,r_i} \leftarrow \mathrm{Com}(i, r_i)$. The issuer signs this commitment, generating $\sigma_i^{id} \leftarrow \mathsf{Schnorr.Sign}_{\mathsf{sk}}(\mathsf{com}_{i,r_i})$, and demonstrates registration with the issuer by producing a proof $\pi_{\mathrm{isu}} := \mathrm{Groth16.Prove}(\mathrm{crs}_{\mathrm{isu}}, (i), (\sigma_i^{id}, r_i))$.
3) Each client i generates an anonymous credential $\tilde{i}_\tau = \mathsf{Prf}_i(\xi_\tau)$ using a randomly chosen value $\xi_\tau \leftarrow \{0,1\}^\lambda$ and uses $(\tilde{i}_\tau, \xi_\tau)$ to participate in the aggregation process, verifying its registration with the issuer through the generated proof.

Blacklist Prevention

1) Server broadcasts the blacklists $\{\mathbf{blk}_\tau^z\}_{z=1}^q$ to all clients $\tilde{i}_\tau$, then updates $\tau \leftarrow \tau + 1$ and sets $\{\mathbf{blk}_\tau^z = \emptyset\}_{z=1}^q$.
2) If $\mathbf{blk}_{\tau-1}^q \neq \emptyset$, clients participating in this round must prove they are not listed in the updated blacklist $\mathbf{blk}_\tau^q$. Each client i generates a proof for the hidden common input i with $\hat{\pi}_{\mathrm{isu}} \leftarrow \textsc{Reg.Cli.}(\mathsf{ck}, \sigma_i^{id}, r_i, i, R_{\mathrm{isu}})$. Client i proves they are not in the $\mathsf{blk}_{\tau-1}^q$ using Groth16 by computing $\pi_{\mathrm{blk}} \leftarrow \mathrm{Groth16.Prove}(\mathrm{crs}_{\mathrm{blk}}, (i, \mathbf{blk}_\tau^q))$.
3) Each client i updates its anonymous credential $\tilde{i}_\tau = \mathsf{Prf}_i(\xi_\tau)$ using a random value ξ_τ. The client generates a series of proofs: $\pi_{\mathrm{anc}} \leftarrow \mathrm{Groth16.Prove}(\mathrm{crs}_{\mathrm{anc}}, (i, \tilde{i}_\tau, \xi_\tau))$, demonstrating that $\tilde{i}_\tau$ is derived from the registered ID i. Additionally, HICIAP is used to generate further proofs: $\hat{\pi}_{\mathrm{isu}} \leftarrow \mathrm{HICIAP.Prove}((\mathsf{ck}, \mathrm{crs}_{\mathrm{isu}}), (i, \pi_{\mathrm{isu}}))$, $\hat{\pi}_{\mathrm{anc}} \leftarrow \mathrm{HICIAP.Prove}((\mathsf{ck}, \mathrm{crs}_{\mathrm{anc}}), (\tilde{i}_\tau, \xi_\tau))$, and $\hat{\pi}_{\mathrm{blk}} \leftarrow \mathrm{HICIAP.Prove}((\mathsf{ck}, \mathrm{crs}_{\mathrm{blk}}), (i, \{\pi_{\mathrm{blk}_z}\}_{z=0}^{\tau-1}))$. The client then links these proofs with $\pi_{\mathrm{link}} \leftarrow \mathrm{HICIAP.LinkProve}(i, \{\hat{\pi}_{\mathrm{isu}}, \hat{\pi}_{\mathrm{anc}}, \hat{\pi}_{\mathrm{blk}}\})$ and sends $(\pi_{\mathrm{zkbl}}, \tilde{i}_\tau, \xi_\tau)$ to the server.
4) The server verifies each client's proof by preparing verification parameters $\hat{\mathbf{S}}_{\mathrm{blk}} \leftarrow \mathrm{Groth16.Prepare}(\mathrm{crs}_{\mathrm{blk}}, \mathsf{blk}_\tau)$, $\hat{\mathbf{S}}_{\mathrm{anc}} \leftarrow \mathrm{Groth16.Prepare}(\mathrm{crs}_{\mathrm{anc}}, (\tilde{i}_\tau, \xi_\tau))$, and $\hat{\mathbf{S}}_{\mathrm{isu}} \leftarrow \mathrm{Groth16.Prepare}(\mathrm{crs}_{\mathrm{isu}})$. Server also generates commitment $\mathsf{com}_{\mathsf{zkbl}} := \mathsf{HICIAP.Com}(\mathsf{ck}, \{\mathsf{ZKB}_z\}_{z=0}^{\tau-1})$. Verification is performed by checking $\mathrm{HICIAP.LinkVfy}(\pi_{\mathrm{link}}, \{\hat{\pi}_{\mathrm{isu}}, \hat{\pi}_{\mathrm{anc}}, \hat{\pi}_{\mathrm{blk}}\})$ and $\mathrm{HICIAP.Vfy}(\mathrm{srs}, \hat{\pi}_{\mathrm{isu}}, \{\hat{\mathbf{S}}_{\mathrm{isu}}\})$. Clients failing verification are prohibited.

Commitments, Distributed Graph Generation, Seed Sharing, Masking stages are not changed

Dropout Agreement and Unmasking (We only marked the changed protocols, and omitted the unchanged parts.)

1) No change.
2) No change.
3) Add: Add the failed clients' anonymous credential $\tilde{i}_\tau$ to the blacklist blk_τ, if any of them fails.
4) No change.
5) Add: Ask clients to give the proof of keys by $\pi_{s_{ij}}$ if verification fails. Add the failed clients' anonymous credential $\tilde{i}_\tau, \xi_\tau$ to the blacklist blk and remove all the keys of malicious clients. Finally, if the $|\mathsf{blk}| = 0$, the server outputs $\sum_{\tilde{i}_\tau \in \mathcal{S}} \mathbf{x}_{\tilde{i}_\tau}$ as $\mathbf{G}^{-1}(\mathrm{Decode}(\mathbf{sk}, \sum_{\tilde{i}_\tau \in A_2'} \mathbf{y}_{\tilde{i}_\tau}))$, else, it go to "Removal of Malicious Clients".

Removal of Malicious Clients

1) For each client $\tilde{m}_\tau$ in the blacklist blk, determine the blacklist level p at which its identity m was initially added. This is done by having each client prove they are not in any specific blacklist level p, as outlined in the Blacklist Prevention step. Once the appropriate level p is identified, update the set $\mathsf{blk}_\tau^p \leftarrow \mathsf{blk}_\tau^p \cup (\tilde{m}_\tau, \xi_\tau)$.
2) The server identifies malicious clients $\tilde{m}_\tau \in \mathsf{blk}$ to generate a seed removal list $\mathsf{R}_{\mathrm{seed}} = \{\tilde{m}_\tau\} \in \mathsf{blk}$.
3) The server then removes the inputs and secret key components linked to the malicious clients from round τ to obtain the final result $\mathbf{a}^* = \textsc{RemoveInput}(\mathbf{a}, \mathsf{blk}, \mathsf{R}_{\mathrm{seed}}, \cdot)$. Go back to **Blacklist Prevention**.