SNARKBlock: Federated Anonymous Blocklisting from Hidden Common Input Aggregate Proofs

Michael Rosenberg*
Department of Computer Science,
University of Maryland
micro@umd.edu

Mary Maller
Ethereum Foundation
mary.maller@ethereum.org

Ian Miers
Department of Computer Science,
University of Maryland
imiers@umd.edu

***Abstract*—Zero-knowledge blocklists allow cross-platform blocking of users but, counter-intuitively, do not link users identities inter- or intra-platform, or to the fact they were blocked. Unfortunately, existing approaches (Tsang et al. '10) require that servers do work linear in the size of the blocklist for each verification of a non-membership proof.**

We design and implement SNARKBLOCK, a new protocol for zero-knowledge blocklisting with server-side verification that is logarithmic in the size of the blocklist. SNARKBLOCK is also the first approach to support ad-hoc, federated blocklisting: websites can mix and match their own blocklists from other blocklists and dynamically choose which identity providers they trust.

Our core technical advance, of separate interest, is the HICIAP zero-knowledge proof system, which addresses a common problem in privacy-preserving protocols: using zero-knowledge proofs for repeated but unlinakble interactions. Rerandomzing a Groth16 proof achieves unlinkability without the need to recompute the proof for every interaction. But this technique does not apply to applications where each interaction includes multiple Groth16 proofs over a common hidden input (e.g., the user's identity). Here, the best known approach is to commit to the hidden input and feed it to each proof, but this creates a persistent identifier, forcing recomputation. HICIAP resolves this problem by aggregating n Groth16 proofs into one $O(\log n)$-sized, $O(\log n)$-verification time proof which also shows that the input proofs share a hidden input. Because HICIAP is zero-knowledge, repeated shows of the same aggregate or an updated aggregate are unlinkable even though the underlying Groth16 proofs are never recomputed.

I. Introduction

Moderation is a powerful tool for combating online harassment, trolling and spam messages. But banning an account on one platform has an obvious problem: it leaves the user free to post under other accounts and on other platforms. As a result, moderation tends towards stronger centralized *identity providers* (e.g., Facebook's real-name policy [Fac]) and the linking of disparate pseudonymous identities within and across platforms. Tying users' online speech to a centralized identity provider poses major problems for the decentralized web and user privacy, and can have a chilling effect on free speech.

*Work supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

Providing both privacy and moderation is a challenge: a *user* posting anonymously on a forum presents two problems to the forum operator, termed *service provider*: access control and revocation. First, because the user's identity is unknown at post submission, the service provider cannot verify that the user is authorized to post (i.e., isn't blocked). Second, because the user's identity is not linked to the post, and posts are not linked together, the service provider cannot revoke the user's posting permissions (i.e., block the user) if their current post violates forum policies. Linking posts together raises privacy concerns that may be undesirable on a single forum and are intolerable if applied across the web.

A. Zero-knowledge proofs of blocklist non-membership

BLAC [TAKS10] introduces the first solution to anonymous blocklisting without a trusted third party. It provides users with long-term identities and allows them to prove, in zero-knowledge, that they are not on a blocklist.

The approach introduced by BLAC, which we formalize as a *zero-knowledge blocklist* (ZKBL), is conceptually simple. A user's identity is a random PRF key k signed by an identity provider to ensure Sybil resistance. Anonymous comments and posts are associated with a tag $\mathsf{tag} := \mathsf{Prf}_k(\mathsf{nonce})$. A blocklist $\mathscr{L}$ consists of tuples $(\mathsf{tag}, \mathsf{nonce})$ from offending posts. A user *attests* that they are not blocked by presenting a fresh $(\mathsf{tag}, \mathsf{nonce})$ pair and a zero-knowledge proof that 1) tag is computed correctly; 2) k is signed by a valid party; and 3) none of the blocklisted tags were generated by k, i.e., $\forall(\mathsf{tag}', \mathsf{nonce}') \in \mathscr{L} : \mathsf{Prf}_k(\mathsf{nonce}') \neq \mathsf{tag}'$. A user is blocked by placing an offending $(\mathsf{tag}, \mathsf{nonce})$ pair on $\mathscr{L}$.

At its core, a ZKBL is a specialized zero-knowledge proof on the PRF evaluation, tag inequalities, and identity signature. Both security and privacy depend, mainly, on the zero-knowledge proof. This gives ZKBLs their main advantage: because the proofs are over arbitrary ban lists, the system is ad-hoc. We do not need a central party to coordinate bans as in [BCD+17], [CL02], [LLX07], [VB20], or worse, a trusted third party who can deanonymize users [Cha85], [Cv91], [BMW03]. If ZKBLs also support private federated identity, this is a major advantage for deployment.

B. Existing ZKBLs are impractical for both clients and servers

Unfortunately, the current approach for ZKBLs [TAKS10] requires the server to do linear work in the size of the blocklist when verifying a non-membership attestation. If the size of the blocklist and the number of attestations per second is proportional to the number of total users, then the service-provider's workload grows *quadratically* as their site scales. This is costly under normal circumstances and can be a major denial of service vector if an attacker can make concurrent posts or obtain Sybil accounts that are later banned.

Almost as problematically, proof sizes are also linear in the size of the blocklist. At 144B per list entry, a single non-membership proof for a 4M-entry blocklist would require a client to upload 549MiB of data over a residential or mobile connection.

C. Our contribution

We design, implement, and benchmark SNARKBLOCK, a protocol for zero-knowledge blocklists which improves on the state of the art by offering log-sized proofs and log-time verification.

Beyond improved performance for ZKBLs, SNARKBLOCK makes ZKBLs fully ad-hoc and resolves a privacy and organization problem with deployment. While ban lists can be operated by anyone, ZKBLs—like any ban system, anonymous or not—require Sybil-resistant identities. Existing ZKBLs assume a single trusted issuer for credentials. In reality, the existence of different issuers will lead to fragmentation of user's identities and also ban lists, reducing anonymity sets and hindering adoption.

SNARKBLOCK removes the need for a single centralized identity provider by allowing service providers to dynamically pick the identity providers that they support. This avoids coordination concerns and allows different providers to adopt different levels of Sybil resistance ranging from CAPTCHAs, to cryptocurrency payment, to real-world identity verification. Crucially, during attestation, the service provider learns only that the user's identity was issued by *some* party in their *accepted identity provider* set.

The core of SNARKBLOCK is a new type of zero-knowledge proof, called HIdden Common Input Aggregate Proofs, or HICIAP (pronounced "high-chop"). HICIAP is a zero-knowledge proof that aggregates n Groth16 [Gro16] proofs (of the same underlying circuit) into a single $O(\log n)$-sized proof, and shows that the aggregated proofs all verify and all share a common input which is not revealed to the verifier. It is also possible to *link* multiple HICIAP proofs, showing in zero-knowledge that their hidden common inputs are all equal. SNARKBLOCK uses HICIAP to aggregate *chunk proofs*—Groth16 proofs of non-membership in equally sized non-overlapping portions of the blocklist.

HICIAP addresses a common problem when using zkSNARKs in privacy-preserving protocols like SNARKBLOCK: repeated interactions can require costly proof recomputations to ensure unlinkability. When presenting a single Groth16 proof, the proof can be rerandomized between interactions, achieving anonymity without recomputaton. Unfortunately, when presenting multiple proofs about a common hidden input—e.g., adding proofs about subsequent state changes—we must recompute all proofs since the state-of-the-art approach is to commit to the hidden input and have all proofs be made with respect to that public commitment. Since the commitment is a persistent identifier, it and all proofs relying on it must be recomputed to achieve anonymity. HICIAP resolves this by supporting zero-knowledge aggregation of proofs with a common hidden input.

II. INTUITION FOR A ZKBL CONSTRUCTION

Zero-knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs), appear to offer a path to ZKBLs with fast verification, but limitations on prover performance—a common problem for nearly all zkSNARKs—make this challenging. This is clear when one examines the costs of using Groth16, a zkSNARK scheme with notably fast verification times.

Existing zkSNARKs can only handle pieces of a blocklist. Producing a zero-knowledge proof of knowledge is, fundamentally, at least linear in the size of the input, i.e., the blocklist. But for Groth16 and other zkSNARKs, the concrete constants are high. Looking ahead, for a blocklist of 256 entries, a single proof of non-membership is 63k constraints and takes 2.84s. A blocklist of 2^{21} entries would yield a proof with 2^{29} constraints. But for Groth16, prover memory usage scales poorly in the size of the circuit: a 2^{29}-constraint proof requires 4TB of memory and takes 3 hours to compute, due to the implementation overhead of distributing proving over a 256-core cluster[WZC+18]. To use zkSNARKs for a ZKBL, we cannot have the prover do work linear in the size of the blocklist for each attestation.

Decomposing blocklists by chunk. We observe that a blocklist, mostly, does not change. While total prover workload is inevitably linear in the size of the blocklist, this work does not need to be recomputed from scratch every time. By breaking the list up into non-overlapping *chunks* we can both reuse work and limit the amount of recomputation required when the list changes.

A zero-knowledge proof for consistency between chunks. A sequence of chunk non-membership proofs for a blocklist $\mathscr{L}$ poses three problems:

1) The server would need to verify $O(|\mathscr{L}|)$ chunks.
2) Reusing a chunk proof across blocklist non-membership attestations would identify the client.
3) A malicious client could use a different identity when proving non-membership in a specific chunk, avoiding a block in that segment of the blocklist.

To address the above problems, we need a *compact* proof that a sequence of chunk non-membership proofs verifies with respect to a single *hidden* identity. Further, that proof must be zero-knowledge to ensure that the chunk proofs can be safely reused across blocklist non-membership proofs.

Recursive Groth16 proofs are impractical. Each chunk proof could recursively check consistency of the previous chunk proof. The challenge for recursion is latency: assuming the blocklist changes frequently, each attestation would need to compute an updated chunk proof and a recursive step. On our benchmark system, a single recursive step for Grooth16 takes 16.5s to prove.[1] With recursion, the cost of computing a chunk proof would also increase by 5–7$\times$ because of overhead imposed by the elliptic curves that are required to support recursion. [CCDW20].[2] Looking ahead, the buffered approach we take adds 1s of latency to attestation and supports a buffer of 14 16-element chunk proofs.

Beyond generic IVC and aggregate proofs. We observe that IVC is not necessary to verify a sequence of chunk non-membership proofs. There is no intermediate state in our computation, rather we only require that all proofs must share the same input private input. Recent advances in inner product proofs [BMM$^+$20] give a succinct proof that n Groth16 zkSNARK proofs verify in aggregate. However, this aggregate approach has two critical shortcomings: it is not zero-knowledge and it does not ensure consistency.

A natural approach for consistency would be to commit to the hidden value and use it as a public input to each Groth16 proof. But if the same commitment is used across multiple anonymous attestations, it forms a persistent identifier. On the other hand, when a fresh commitment is used for each attestation, we must regenerate every chunk proof.

We use [BMM$^+$20] as a starting point and have a single *public* input to each chunk proof, then blind it in the aggregate proof so it is not revealed to the verifier. The resulting scheme reuses the same blinders in multiple parts of the zero-knowledge protocol. This unusual property made proving the honest-verifier zero-knowledge property challenging.

[1]This is the cost to verify a proof with no inputs using MNT6-753 over MNT4-753.

[2]*Depth-1 recursion* using, e.g., BW6-761 over BLS12-377, would avoid some of these costs compared to the MNT4/6 cycle. However, any addition to the blocklist would necessitate recomputing the top-level proof that checks all n chunk proofs. At 3.9s per check on our benchmark system, this is not feasible.

III. Preliminaries

We write $x := z$ to denote variable assignment, and $y \leftarrow S$ to denote sampling uniformly from a set S. For an arbitrary, efficiently computable predicate P, we say that *a proof of knowledge of a relation* $R = \{(\mathbf{x};\mathbf{w}) : P(\mathbf{x},\mathbf{w})\}$ with respect to an *instance* $\mathbf{x}$ is a proof of knowledge of the *witness* $\mathbf{w}$ such that $P(\mathbf{x},\mathbf{w})$ is satisfied. We will often refer to $\mathbf{x}$ as a *public input* and $\mathbf{w}$ as a *private input*, and we will use zero-knowledge proofs of knowledge for various relations in order to hide $\mathbf{w}$ from the verifier. The security parameter of our system is denoted by λ.

A. Notation for Groups and Pairings

We will work exclusively with prime-order groups and their associated scalar fields. Group elements are denoted with capital letters $G \in \mathbb{G}$, while field elements are lowercase $r \in \mathbb{F}$. *Vectors* are bolded: $\mathbf{A} \in \mathbb{G}^n$, and $\mathbf{r} \in \mathbb{F}^n$. We write $\mathbf{A}_{[:k]}$ to denote the first k elements of $\mathbf{A} \in \mathbb{G}^n$, and $\mathbf{A}_{[k:]}$ to denote the last $n-k$ elements. We say that a bilinear function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a *type-3 bilinear pairing* if there is no efficiently computable group homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. We say e is *degenerate* if there is a non-identity $G \in \mathbb{G}_1$ such that $e(G,H) = 1$ for all $H \in \mathbb{G}_2$. Following convention, we use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$, and multiplicative notation for $\mathbb{G}_T$.

For vectors $\mathbf{A} \in \mathbb{G}_1^n$ and $\mathbf{B} \in \mathbb{G}_2^n$ and a bilinear pairing e, we write $\mathbf{A} * \mathbf{B}$ to denote the *inner pairing product* $\prod_{i=1}^n e(A_i, B_i)$. For vectors $\mathbf{A} \in \mathbb{G}^n$ and $\mathbf{r} \in \mathbb{F}^n$ we write $\mathbf{A}^{\mathbf{r}}$ to denote the *multiscalar multiplication* (MSM) $\sum_{i=1}^n r_i A_i$, and write $\mathbf{r} \odot \mathbf{A}$ to denote the element-wise multiplication $(r_1 A_1, \ldots, r_n A_n)$. For a field element $x \in \mathbb{F}$, we denote $[x]_1 := xG$ and $[x]_2 := xH$, where G and H are the canonical generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

B. Groth16

We briefly describe the trusted-setup zkSNARK scheme defined in [Gro16]. At a high level, given a description of an arithmetic circuit (over the scalar field of a pairing-friendly elliptic curve), a Groth16 proof proves that a circuit is satisfied by a set of public wires (values known to the verifier) and private wires (values which are not known to the verifier, also called *witness elements*).

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be an efficiently-computable, non-degenerate, type-3 bilinear pairing, where $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T|$ is a prime p and $p > 2^{\lambda}$. Let G be a generator of $\mathbb{G}_1$ and H be a generator of $\mathbb{G}_2$. We use $\mathbb{F}$ to denote the finite field $\mathbb{Z}/p\mathbb{Z}$. The Groth16 scheme defines four procedures:

$\mathsf{Setup}(\mathsf{desc}) \to \mathsf{crs}$ Generates a common reference string for the given arithmetic circuit description. crs contains the group elements necessary to compute the expressions in $\mathsf{Groth16.Prove}$ below.

$\mathsf{Prove}(\mathsf{crs}, \{a_i\}_{i=0}^{\ell}, \{a_i\}_{i=\ell+1}^{m}) \to \pi$ Proves the circuit described by crs is satisfied, where $a_0, \ldots, a_\ell \in \mathbb{F}$ represent the circuit's public input wires and $a_{\ell+1}, \ldots, a_m \in \mathbb{F}$ represent the private wires. π is of the form $([\eta]_1, [\theta]_2, [\iota]_1)$, where

$$\eta = \alpha + \sum_{i=0}^{m} a_i u_i(X) + r\delta \quad \theta = \beta + \sum_{i=0}^{m} a_i v_i(X) + s\delta$$
$$\iota = \sum_{i=\ell+1}^{m} \frac{a_i\left(\beta u_i(X) + \alpha v_i(X) + w_i(X)\right) + h(X)t(X)}{\delta} + \eta s + \theta r - rs\delta$$

and all otherwise unspecified constants and polynomials come from crs.

$\mathsf{Prepare}(\mathsf{crs}, \{a_{i_j}\}_{j=1}^{t}) \to \hat{S}$ Aggregates any subset of public inputs into a single group element called a *prepared input*: $\hat{S} = \sum_{j=1}^{t} a_{i_j} W_{i_j}$, where W_i are the CRS values whose coefficient represents the value of the i-th wire of the circuit.

$\mathsf{Vfy}(\mathsf{crs}, \pi, \{a_0\}_{i=0}^{\ell}) \to \{0,1\}$ Verifies the proof $\pi = (A, B, C)$ by checking the relation,

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

where $[\alpha]_1$, $[\beta]_2$, $[\gamma]_2$, and $[\delta]_2$ come from crs. Vfy permits any subset of the public inputs to be prepared as above. The common case will be where all but the first input is prepared, i.e., calls of the form $\mathsf{Vfy}(\mathsf{crs}, \pi, (a_0, \hat{S}))$.

$\mathsf{Rerand}(\pi) \to \pi'$ Rerandomizes the proof $\pi = (A, B, C)$ by sampling $\zeta, \omega \leftarrow \mathbb{F}$ and computing

$$\pi' := (\zeta^{-1} A, \zeta B + \zeta\omega [\delta]_2, C + \omega A).$$

By Theorem 3 in [BKSV20], the output of Rerand is statistically indistinguishable from a fresh proof of the same underlying statement.

C. Inner product proofs

Bünz et al. [BMM+20] introduce a proof system for various inner product relations. We will make use of

$$R_{\mathsf{TIPP}} := \left\{ \begin{array}{ll} \multicolumn{2}{c}{\begin{pmatrix} \mathsf{ck}_1, \in \mathbb{G}_2^n, \mathsf{ck}_2 \in \mathbb{G}_1^n, \\ \mathsf{com}_A, \mathsf{com}_B, \mathsf{agg}_{AB} \in \mathbb{G}_T, \\ \mathbf{r} \in \mathbb{F}^n; \mathbf{A} \in \mathbb{G}_1^n, \mathbf{B} \in \mathbb{G}_2^n \end{pmatrix} :} \\ & \mathsf{com}_A = \mathbf{A} * \mathsf{ck}_1 \\ \bigwedge & \mathsf{com}_B = \mathsf{ck}_2 * \mathbf{B} \\ \bigwedge & \mathsf{agg}_{AB} = \mathbf{A}^{\mathbf{r}} * \mathbf{B} \end{array} \right\}$$

$$R_{\mathsf{MIPP}-k} := \left\{ \begin{array}{c} \begin{pmatrix} \mathsf{ck}_1 \in \mathbb{G}_2^n, \\ \mathsf{com}_C, \in \mathbb{G}_T, \mathsf{agg}_C \in \mathbb{G}_1, \\ \mathbf{r} \in \mathbb{F}^n; \mathbf{C} \in \mathbb{G}_1^n \end{pmatrix} : \\ \mathsf{com}_C = (\mathbf{C} * \mathsf{ck}_1) \bigwedge \mathsf{agg}_C = \mathbf{C}^{\mathbf{r}} \end{array} \right\}$$

$$R_{\mathsf{HMIPP}} := \left\{ \begin{array}{ll} \multicolumn{2}{c}{\begin{pmatrix} \mathsf{ck}_1 \in \mathbb{G}_2^n, \mathsf{ck}_3 \in \mathbb{G}_2, \\ \mathsf{com}_C, \in \mathbb{G}_T, \mathsf{agg}_C \in \mathbb{G}_1, \\ \mathbf{r} \in \mathbb{F}^n; \mathbf{C} \in \mathbb{G}_1^n, z \in \mathbb{F} \end{pmatrix} :} \\ & \mathsf{com}_C = e([z]_1, \mathsf{ck}_3) \cdot (\mathbf{C} * \mathsf{ck}_1) \\ \bigwedge & \mathsf{agg}_C = \mathbf{C}^{\mathbf{r}} \end{array} \right\}$$

Fig. 1: We directly use Bünz et al.'s definition of R_{TIPP} and $R_{\mathsf{MIPP}\text{-}k}$, and we use R_{HMIPP} to refer to the "hiding commitment" version of $R_{\mathsf{MIPP}\text{-}k}$. While R_{HMIPP} admits a zero-knowledge proof of knowledge, R_{TIPP} does not, as it fails to hide the witnesses $\mathbf{A}$ and $\mathbf{B}$. Patching this is one of the primary focuses of HICIAP.

the TIPP, MIPP_k, and HMIPP proof systems, whose relations are defined in Figure 1.

In short, R_{TIPP} is satisfied when $\mathbf{A}^{\mathbf{r}} * \mathbf{B} = \mathsf{agg}_{AB}$, $R_{\mathsf{MIPP}-k}$ is satisfied when $\mathbf{C}^{\mathbf{r}} = \mathsf{agg}_C$, and R_{HMIPP} is the same as $R_{\mathsf{MIPP}-k}$ except its commitment to $\mathbf{C}$ is hiding.

D. HICIAP

Since HICIAP is used extensively in the construction of SNARKBLOCK, we provide a brief overview of its functionality here. We defer discussion of these algorithms including their construction and security claims until Section VI.

HICIAP is a zkSNARK which aggregates multiple Groth16 proofs of the same relation. Of its aggregated proofs, it proves that 1) they verify with respect to verifier-supplied public inputs, and 2) they share a common public input element (which is hidden by the aggregate proof). In addition, HICIAP can *link* aggregate proofs: it can prove in zero-knowledge that the proofs in a set of HICIAP proofs all share the same common input element.

Formally, HICIAP consists of six procedures:

$\mathsf{GenCk}(n) \to (\mathsf{ck}, \mathsf{srs})$ Generates a commitment key $(\mathsf{ck}_1, \mathsf{ck}_2, \mathsf{ck}_3) \in \mathbb{G}_2^n \times \mathbb{G}_1^n \times \mathbb{G}_2$ and a (short) structured verification key srs which can be used, respectively, to prove and verify HICIAP aggregates of up to $n-2$ Groth16 proofs, where n is a power of 2.

$\mathsf{Com}(\mathsf{ck}, \hat{\mathbf{S}}) \to \mathsf{com}_{\mathsf{in}}$ Constructs a commitment to the prepared Groth16 public inputs $\hat{\mathbf{S}} \in \mathbb{G}_1^{n-2}$ as $\mathsf{com}_{\mathsf{in}} := \hat{\mathbf{S}} * \mathsf{ck}_{1,[:n-2]}$.

$\mathsf{Prove}((\mathsf{ck}, \mathsf{crs}), \hat{\mathbf{S}}, (a_0, \{\pi_i\}_{i=1}^{n-2})) \to (\hat{\pi}, o)$ Produces a succinct proof that each Groth16 proof π_i verifies w.r.t. the common witness element $a_0 \in \mathbb{F}$, the prepared input $\hat{S}_i \in \mathbb{G}_1$, and the given Groth16 CRS. Also produces an opening o to a commitment to a_0 contained inside $\hat{\pi}$. The opening is used in LinkProve.

$\mathsf{Vfy}(\mathsf{srs}, \hat{\pi}, \mathsf{com}_{\mathsf{in}}) \to \{0,1\}$ Verifies the given aggregate proof w.r.t. the committed public input. Alternatively, a set of prepared Groth16 inputs can be passed instead of $\mathsf{com}_{\mathsf{in}}$.

$\mathsf{LinkProve}(\{\hat{\pi}_i\}_{i=1}^{t}, (a_0, \{o_i\}_{i=1}^{t})) \to \pi_{\mathsf{link}}$ Using the openings o_i, produces a proof that the given aggregate proofs share the witness element $a_0 \in \mathbb{F}$.

$\mathsf{LinkVfy}(\pi_{\mathsf{link}}, \{\hat{\pi}_i\}_{i=1}^{t}) \to \{0,1\}$ Verifies the link proof w.r.t. the given aggregate proofs.

IV. Zero-knowledge blocklists

We now give our framework for zero-knowledge blocklists, taken directly from BLAC [TAKS10], but with modifications to support multiple identity providers and allow for additional precomputation.

A. Setting

A zero-knowledge blocklist allows users to attest that an *identity* issued by one of a set of *identity providers* is not in a *blocklist*. We now detail these concepts:

Identity. We use k to denote a user's private *identity*. A single user in the real world can hold arbitrarily many identities. In all cases, k will be a field element selected uniformly at random by the user. Other similar schemes refer to k as a user's "nym," "pseudonym," or "credential."

Identity providers. Blocking users fundamentally depends on identities being Sybil-resistant. Most approaches to blocklisting, including BLAC's approach to ZKBLs, assume a single issuer.

Here we formalize a more general version of ZKBLs that supports federated identity:each service provider is allowed to maintain its own list $\mathscr{I}$ of accepted identity providers, which we call the *AIP set*. Identity providers are responsible for ensuring Sybil resistance. The service provider is allowed to update this set over time, and should distribute it via the same channels it uses to distribute its blocklist.

Blocklists and session tuples. A ZKBL *blocklist* consists of pairs containing a *session nonce* nonce and *session tag* tag, where tag is bound to the user's identity by $\mathsf{tag} := \mathsf{Prf}_k(\mathsf{nonce})$ for some fixed pseudorandom function Prf. Blocklist entries can support *context binding* via structured auxiliary data. By computing nonce as $\mathsf{nonce} := \mathsf{H}(\mathsf{aux}, r)$ for some hash H, aux is bound to the attestation. This data can be used to, for example, bind attestation to an action (e.g., to prove that the blocked user is the action's author) or to a particular blocklist or policy (e.g., to enforce which lists a banned tuple can be transferred to).

Finally, in a departure from BLAC, we allow blocklists to be split into *chunks*—equally sized non-overlapping segments—whose sizes are decided by the service provider. Blocklists are chunked so that users can precompute non-membership proofs over individual chunks rather than the entire blocklist at once.

Formalizing non-membership proofs. A non-membership proof π_{zkbl} is a zero-knowledge proof of three distinct properties:

1) *Issuance.* That the user's identity k is signed by an identity provider.
2) *Tag well-formedness.* That tag and nonce are honestly computed, i.e., $\mathsf{tag} = \mathsf{Prf}_k(\mathsf{nonce})$.
3) *Blocklist non-membership.* That the user's identity k did not generate any tuples already on a blocklist, i.e., $\forall(\mathsf{tag}', \mathsf{nonce}') \in \mathscr{L} : \mathsf{tag}' \neq \mathsf{Prf}_k(\mathsf{nonce}')$.

B. ZKBL functionality

A zero-knowledge blocklist consists of five algorithms.

CRS-Setup Generates system-wide parameters.

IdP-Keygen Generates a signing keypair $(\mathsf{sk}, \mathsf{pk})$ to be used by an identity provider for issuance.

Register Executes a protocol between a user and an identity provider. On success, the user obtains a signature of (a function of) their identity.

Sync Fetches the latest additions to a service provider's blocklist and then precomputes cryptographic material for them. Users periodically run this routine *offline*, i.e., when not attesting.

Attest Executes a non-interactive protocol in which a user authenticates to a service provider. First, the user constructs a session-specific tuple $(\mathsf{tag}, \mathsf{nonce})$ as $\mathsf{tag} := \mathsf{Prf}_k(\mathsf{nonce})$, where nonce is pseudorandom and optionally bound a context aux. This tuple can be used by the service provider to block the user at any point in the future by simply including it in the blocklist. The user then produces a zero-knowledge proof π_{zkbl} that proves well-formedness of the tuple and that their (signed) identity did not generate any tuples already on a blocklist. The session tuple and zero-knowledge proof are then sent to the service provider as $(\pi_{\mathsf{zkbl}}, \mathsf{tag}, \mathsf{nonce})$.

Verify Checks the validity of a user's attestation. A service provider accepts if and only if π_{zkbl} verifies with respect to the supplied session tuple $(\mathsf{tag}, \mathsf{nonce})$

and the service provider's blocklist $\mathscr{L}$, chunk size schedule, AIP set $\mathscr{I}$, and optional context-binding string aux.

Separately, we assume two non-cryptographic operations for blocklist management:

Blocklist-Add Adds a token to a blocklist.

Blocklist-Remove Removes a token from a blocklist.

We stress that the Add and Remove routines are distinct from the cryptographic scheme, and can be run by anyone. How parties decide to manage their blocklists is wholly orthogonal to the ZKBL construction.

BLAC as a ZKBL. The authors of BLAC construct their scheme using BBS+ signatures [BBS04] and a Camenisch-Shoup Σ-protocol [CS03]. Although not described as such, this is the same PRF approach we formalize here. BLAC's tag function is $\mathsf{nonce} \mapsto \mathsf{H}(\mathsf{nonce})^k$, and it is done in two steps, with the hash evaluation outside the zero-knowledge proof, and the exponentiation witnessed inside. Conceptually, the entire question for designing a practical ZKBL is how to co-design a PRF and zero-knowledge proof protocol to make an efficient non-membership proof.

C. Security requirements

Our desired security properties are taken from BLAC. For the complete definitions see [TAKS10]. Note the following aesthetic changes in our description: blocklistability encompasses misauthentication resistance; and anonymity is described as a distinguishability notion as opposed to a simulatability notion, which we believe better captures the actual security properties achieved by BLAC's game-based definition.

Blocklistability A coalition of dishonest service providers and users can only successfully authenticate to an honest service provider if that user holds a valid credential issued by an identity provider that is not included in the blocklist.

Non-Frameability A coalition of dishonest identity providers, service providers, and users cannot prevent an honest, non-blocklisted user from successfully authenticating with an honest service provider.

Anonymity A coalition of dishonest identity providers, service providers, and users cannot distinguish attestation transcripts associated with any two honest users. Further, no such coalition can link any given authentication transcript with the registration in which an identity provider issued the associated credential.

V. SNARKBLOCK DESIGN AND OVERVIEW

The full design of SNARKBLOCK is detailed in Figure 3. The core relations are defined in Figure 2. In words, R_{isu} is satisfied when a user's committed identity is signed by an issuer in the AIP set, R_{tag} is satisfied when tag is computed correctly, and R_{chunk} is satisfied when a user did not produce any of the tags in a chunk.

$$R_{\mathsf{isu}} := \left\{ \begin{array}{c} (k, (\mathsf{pk}_i)_{i=1}^{\ell}\,; i^*, \sigma, r) : \\ 1 \le i^* \le \ell \\ \bigwedge \mathsf{Schnorr.Ver}_{\mathsf{pk}_{i^*}}(\mathsf{Com}(k,r), \sigma) \end{array} \right\}$$

$$R_{\mathsf{tag}} := \{(k, \mathsf{tag}, \mathsf{nonce}) : \mathsf{Prf}_k(\mathsf{nonce}) = \mathsf{tag}\}$$

$$R_{\mathsf{chunk}} := \left\{ (k, \mathsf{chunk}) : \bigwedge_{(\mathsf{tag},\mathsf{nonce}) \in \mathsf{chunk}} \mathsf{Prf}_k(\mathsf{nonce}) \ne \mathsf{tag} \right\}$$

$$R_{\mathsf{zkbl}} := \left\{ \begin{array}{cc} & (\mathscr{L}, \mathscr{I}, \mathsf{tag}, \mathsf{nonce}\,; k, i^*, \sigma, r) : \\ & R_{\mathsf{isu}}(k, \mathscr{I}\,; i^*, \sigma, r) \\ \bigwedge & R_{\mathsf{tag}}(k, \mathsf{tag}, \mathsf{nonce}) \\ \bigwedge_{i=1}^{c} & R_{\mathsf{chunk}}(k, \mathsf{chunk}_i) \end{array} \right\}$$

Fig. 2: R_{zkbl} is the relation which the attestation procedure in SNARKBLOCK attests to. $\mathscr{I}$ is the AIP set $\{\mathsf{pk}_1, \dots, \mathsf{pk}_\ell\}$, and $\mathscr{L}$ is the set of chunks $\{\mathsf{chunk}_1, \dots, \mathsf{chunk}_c\}$. Note that k is a public (rather than private) input to the three sub-relations R_{isu}, R_{tag}, and R_{chunk}. This is because the implementation of HICIAP requires that the hidden common input be a public input in the underlying Groth16 proof.

We omit textual descriptions of the full set of algorithms and detail the two key ones: Sync and Attest.

Sync. Sync is the offline phase of attestation. During Sync a client fetches the most recent versions of the service provider's blocklist, chunk schedule, and AIP set. The client then precomputes Groth16 chunk proofs π_{chunk_i} of the relation $R_{\mathsf{chunk}}(k, \mathsf{chunk}_i)$ for every new chunk_i received from the service provider. The client also precomputes π_{isu}, by computing a Groth16 proof π_{isu} of $R_{\mathsf{isu}}((k, \mathscr{I}), (i^*, \sigma, r))$ where i^* is the chosen identity provider in the AIP set $\mathscr{I} = \{\mathsf{pk}_1, \dots, \mathsf{pk}_\ell\}$, σ is the identity provider's signature of the identity commitment, and r is the randomness used to commit to k.

Attest. To attest to blocklist non-membership, the client must combine a series of proofs about the user's identity k. First the client computes fresh session tuple $(\mathsf{tag}, \mathsf{nonce})$ and proves it is well-formed with respect to k using a Groth16 proof π_{tag} for the relation $R_{\mathsf{tag}}(k, \mathsf{tag}, \mathsf{nonce})$.

Ideally, the client would combine π_{tag} with the precomputed π_{isu} and π_{chunk_i} proofs from Sync. But a single HICIAP instance only works for proofs over the same relation. Thus, π_{tag} and π_{isu} are wrapped in HICIAP proofs $\hat{\pi}_{\mathsf{tag}}$ and $\hat{\pi}_{\mathsf{isu}}$ respectively, the π_{chunk_i} proofs are aggregated into a HICIAP proof $\hat{\pi}_{\mathsf{chunk}}$, and a linking proof π_{link} is used to show each aggregate is made with

```
$\underline{\mathsf{IdPKeyGen}()}$
$(\mathsf{sk},\mathsf{pk}) := \mathsf{Schnorr.KeyGen}()$
return $(\mathsf{sk},\mathsf{pk})$

$\underline{\mathsf{RegU}(k)}$
$r \leftarrow \mathbb{F}$
$\mathsf{com} := \mathsf{Com}(k,r)$
return $(\mathsf{com},r)$

$\underline{\mathsf{RegS}(\mathsf{sk},\mathsf{com})}$
$\sigma := \mathsf{Schnorr.Sign}_{\mathsf{sk}}(\mathsf{com})$
return $\sigma$

$\underline{\mathsf{CrsSetup}(n)}$
$\mathsf{crs}_{\mathsf{isu}} := \mathsf{Groth16.Setup}(R_{\mathsf{isu}})$
$\mathsf{crs}_{\mathsf{tag}} := \mathsf{Groth16.Setup}(R_{\mathsf{tag}})$
$\mathsf{crs}_{\mathsf{chunk}} := \mathsf{Groth16.Setup}(R_{\mathsf{chunk}})$
$(\mathsf{ck},\mathsf{srs}) := \mathsf{HICIAP.GenCk}(n)$
return $(\mathsf{ck},\mathsf{srs})$

$\underline{\mathsf{Attest}(k,\pi_{\mathsf{isu}},\{\pi_{\mathsf{chunk}_i}\}_{i=1}^{c})}$
$\mathsf{nonce} \leftarrow \{0,1\}^{\lambda}$
$\mathsf{tag} := \mathsf{Prf}_k(\mathsf{nonce})$
$\pi_{\mathsf{tag}} := \mathsf{Groth16.Prove}(\mathsf{crs}_{\mathsf{tag}},(k,\mathsf{tag},\mathsf{nonce}),\cdot)$
$\hat{\pi}_{\mathsf{isu}} := \mathsf{HICIAP.Prove}((\mathsf{ck},\mathsf{crs}_{\mathsf{isu}}),\mathscr{I},(k,\{\pi_{\mathsf{isu}}\}))$
$\hat{\pi}_{\mathsf{tag}} := \mathsf{HICIAP.Prove}((\mathsf{ck},\mathsf{crs}_{\mathsf{tag}}),(\mathsf{tag},\mathsf{nonce}),(k,\{\pi_{\mathsf{tag}}\})))$
$\hat{\pi}_{\mathsf{chunk}} := \mathsf{HICIAP.Prove}\begin{pmatrix} (\mathsf{ck},\mathsf{crs}_{\mathsf{chunk}}),\mathscr{L}, \\ (k,\{\pi_{\mathsf{chunk}_i}\}_{i=1}^{c}) \end{pmatrix}$
$\pi_{\mathsf{link}} := \mathsf{HICIAP.LinkProve}(k,(\pi_{\mathsf{isu}},\pi_{\mathsf{chunk}},\pi_{\mathsf{tag}}),k)$
$\pi_{\mathsf{zkbl}} := (\pi_{\mathsf{link}},\hat{\pi}_{\mathsf{isu}},\hat{\pi}_{\mathsf{tag}},\hat{\pi}_{\mathsf{chunk}})$
return $(\pi_{\mathsf{zkbl}},\mathsf{tag},\mathsf{nonce})$

$\underline{\mathsf{Sync}(\{\mathsf{chunk}_i\}_{i=c'}^{c},\mathscr{I},i^*,k,\sigma,r)}$
for $c' \le j \le c$ :
    $\pi_{\mathsf{chunk}_j} := \mathsf{Groth16.Prove}(\mathsf{crs}_{\mathsf{chunk}},(k,\mathsf{chunk}_j),\cdot)$
$\pi_{\mathsf{isu}} := \mathsf{Groth16.Prove}(\mathsf{crs}_{\mathsf{isu}},(k,\mathscr{I}),(i^*,\sigma,r))$
return $\{\pi_{\mathsf{chunk}_1},\ldots,\pi_{\mathsf{chunk}_c}\}$

$\underline{\mathsf{PrepBlocklist}(\{\mathsf{chunk}_i\}_{i=1}^{c})}$
for $1 \le i \le c$
    $\hat{S}_{\mathsf{chunk}_i} := \mathsf{Groth16.Prepare}(\mathsf{crs}_{\mathsf{chunk}},\mathsf{chunk}_i)$
$\mathsf{com}_{\mathscr{L}} := \mathsf{HICIAP.Com}(\mathsf{ck},\{\hat{S}_{\mathsf{chunk}_i}\}_{i=1}^{c})$
return $\mathsf{com}_{\mathscr{L}}$

$\underline{\mathsf{Vfy}(\pi_{\mathsf{zkbl}},(\mathsf{tag},\mathsf{nonce}),\mathscr{I},\mathsf{com}_{\mathscr{L}})}$
$\hat{S}_{\mathsf{tag}} = \mathsf{Groth16.Prepare}(\mathsf{crs}_{\mathsf{tag}},(\mathsf{tag},\mathsf{nonce}))$
$\hat{S}_{\mathsf{isu}} = \mathsf{Groth16.Prepare}(\mathsf{crs}_{\mathsf{isu}},\mathscr{I})$
return $\mathsf{HICIAP.LinkVfy}(\pi_{\mathsf{link}},(\hat{\pi}_{\mathsf{isu}},\hat{\pi}_{\mathsf{tag}},\hat{\pi}_{\mathsf{chunk}}))$
$\bigwedge$    $\mathsf{HICIAP.Vfy}(\mathsf{srs},\hat{\pi}_{\mathsf{isu}},\{\hat{S}_{\mathsf{isu}}\})$
$\bigwedge$    $\mathsf{HICIAP.Vfy}(\mathsf{srs},\hat{\pi}_{\mathsf{tag}},\{\hat{S}_{\mathsf{tag}}\})$
$\bigwedge$    $\mathsf{HICIAP.Vfy}(\mathsf{srs},\hat{\pi}_{\mathsf{chunk}},\mathsf{com}_{\mathscr{L}})$
```

Fig. 3: A pseudocode definition of the SNARKBLOCK system. We omit buffering and nonce binding.

respect to the same identity k.

The client's output is thus $(\pi_{\mathsf{zkbl}},\mathsf{tag},\mathsf{nonce})$, where $\pi_{\mathsf{zkbl}} := (\hat{\pi}_{\mathsf{isu}},\hat{\pi}_{\mathsf{tag}},\hat{\pi}_{\mathsf{chunk}},\pi_{\mathsf{link}})$.

Buffering recent blocklist additions and deletions. When a ban is added or removed from the blocklist, the user must redo the corresponding chunk proof. It is inevitable between Sync operations that some number of additions and deletions will occur, thus requiring recomputation during Attest and adding the corresponding amount of latency. The larger the chunk size, the higher the latency. While we can avoid this for deletions by having bans expire in batches, this is undesirable for additions—we want bans to take effect as soon as possible.

To avoid a tradeoff between chunk size and attestation latency, we have the tail of the list be a buffer of smaller chunks and have a separate instance of HICIAP aggregate them. Because the circuit is different from the circuit used for larger chunks, this optimization increases the number of distinct HICIAP proofs passed to the verifier, while decreasing the overall attestation time.

A. Security argument

Security of SNARKBLOCK depends on it correctly instantiating the PRF+Sig+ZKP paradigm using HICIAP. We state the theorem of security for SNARKBLOCK here and give a proof sketch in Appendix B. This proof depends on the security of HICIAP as a building block, and so HICIAP is the main focus of our security analysis over subsequent sections and appendices.

Theorem 1 (SNARKBLOCK Security)**.** SNARKBLOCK *described in Figure 3 is blocklistable, anonymous and non-frameable provided that Groth16 and* HICIAP *proofs are knowledge-sound and subversion zero-knowledge; Schnorr signatures are unforgeable;* Prf *is pseudorandom; and* Com *is binding and hiding.*

Looking ahead, in the concrete instantiation, this in turn assumes the key-prefixed Poseidon hash function is a PRF and, for Groth16, that the q-SDH [BB04] and

q-DDH [BB04] assumptions hold in the Algebraic Group Model [FKL18]. For HICIAP we also depend on the Auxiliary Structured Double Pairing assumption [BMM+20].

B. Trusted setup

Our protocol and security proof assumes that a trusted party generates a CRS for each Groth16 circuit as well as for each HICIAP instance. The CRSs are similar, being of the form $sG, s^2G, \ldots$ for several bases. In most cases, service providers should be able to run the setup: assuming subversion resistance [Fuc17], a malicious CRS only undermines soundness, *not* privacy. If necessary, protocols [BGM17], [BCG+15] for multiparty setup have been used for commercial cryptocurrencies such as Zcash [Rad21] and Filecoin, where failure would allow the forgery of billions of dollars. These also ensure subversion resistance.

VI. HICIAP

We now introduce the core of SNARKBLOCK: HIdden Common-Input Aggregate Proofs (HICIAP), a novel zkSNARK scheme which we use to generate the zero-knowledge proof of blocklist non-membership π_{zkbl}.

Recall the purpose of HICIAP is to aggregate multiple Groth16 proofs of the same relation. Of its aggregated proofs, it proves that 1) they verify with respect to verifier-supplied public inputs, and 2) they share a common public input element (which is hidden by the aggregate proof). In the case of SNARKBLOCK, the relation is chunk non-membership, the verifier-supplied public inputs are the (prepared) blocklist chunks, and the common input element is the user's identity.

In addition, HICIAP can *link* aggregate proofs: it can prove in zero-knowledge that the proofs in a set of HICIAP proofs all share the same common input element. In the case of (unbuffered) SNARKBLOCK, there are three aggregate proofs that are linked: chunk non-membership, issuance, and tag well-formedness.

In this section, we provide intuition for the design of HICIAP and then describe each of its components in detail.

A. Intution

To explain HICIAP, we start with a naïve verifier who takes a full (non-succinct and non-hiding) set of Groth16 proofs $\pi_i = (A_i, B_i, C_i)$ and checks that they verify with respect to a common public input. We transform this into a succinct zero-knowledge proof that vector commitments to $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ contain proofs that verify with respect to a hidden input. For simplicity, we omit the blinding factors from discussion, and leave their detailed accounting to the proof of honest verifier zero-knowledge in Appendix B.

The HICIAP verifier must be convinced that there is some hidden wire value $a_0 \in \mathbb{F}$ for which a set Groth16 proofs $\{(A_i, B_i, C_i)\}_{i=1}^{n-2}$ verify given a set of prepared public inputs $\hat{\mathbf{S}}$, i.e., for all $i = 1\ldots, n-2$,

$$e(A_i, B_i) \stackrel{?}{=} e([\alpha]_1, [\beta]_2) \cdot e(C_i, [\delta]_2) \cdot e\left(a_0 W_0 + \hat{S}_i, [\gamma]_2\right).$$

Our first step is to combine the above $n-2$ equations into a single polynomial equation. Verifying this would still require the verifier to do linearly many equality checks, so the verifier picks a random $r \leftarrow \mathbb{F}$ and evaluates the polynomial "in the exponent" at the random point. Combining these two steps, the new verifier equation is

$$\begin{aligned}&\prod e(A_i, B_i)^{r^i} \\ &\stackrel{?}{=} \prod e([\alpha]_1, [\beta]_2)^{r^i} \cdot \prod e(C_i, [\delta]_2)^{r^i} \\ &\quad \cdot \prod e\left(a_0 W_0 + \hat{S}_i, [\gamma]_2\right)^{r^i}.\end{aligned}$$

By the Schwartz-Zippel lemma, equality here implies the equality of the initial $n-2$ equations with overwhelming probability. We now have one equality check.

The next step is to make the verifier oblivious to a_0. To do that, we split the $e(a_0 W_0 + \hat{S}_i, [\gamma]_2)^{r^i}$ term in two. The prover sends W, a blinded version of $\sum r^i a_0 W_0$, to the verifier. It proves that W is computed correctly using an instance of the Σ-protocol HWW (described in Section VI-B). The verifier equation is now

$$\begin{aligned}&\prod e(A_i, B_i)^{r^i} \\ &\stackrel{?}{=} \prod e([\alpha]_1, [\beta]_2)^{r^i} \cdot \prod e(C_i, [\delta]_2)^{r^i} \cdot e(W, [\gamma]_2) \\ &\quad \cdot \prod e\left(\hat{S}_i, [\gamma]_2\right)^{r^i}.\end{aligned}$$

For both succinctness and privacy, the prover cannot give the verifier every Groth16 proof. Instead, the prover gives only succinct commitments, $\mathsf{com}_A, \mathsf{com}_B, \mathsf{com}_C$ to the proof vectors $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, respectively. This requires the prover to calculate $\mathsf{agg}_{AB} := \prod e(A_i, B_i)^{r^i}$ and $\mathsf{agg}_C := \prod e(C_i, [\delta]_2)^{r^i}$ itself and send them to the verifier. Since these calculations can be expressed as inner product operations, the prover shows they are correct using instances of TIPP and HMIPP, respectively. The verifier equation is now

$$\begin{aligned}&\mathsf{agg}_{AB} \\ &\stackrel{?}{=} \prod e([\alpha]_1, [\beta]_2)^{r^i} \cdot \mathsf{agg}_C \cdot e(W, [\gamma]_2) \cdot \prod e\left(\hat{S}_i, [\gamma]_2\right)^{r^i}.\end{aligned}$$

This equation is now verifiable by the HICIAP verifier, but it is not fully succinct—the verifier must still do linear work in order to compute the products containing the r^i exponents. The verifier can avoid this for the term $\prod e([\alpha]_1, [\beta]_2)^{r^i}$ by simply using the geometric sum formula: $\sum_{i=0}^{n} r^i = (r^{n+1} - 1)/(r-1)$. The second optimization, due to Bünz et al. [BMM+20], moves the aggregation of the prepared inputs $\mathsf{agg}_{\mathsf{in}} := \sum r^i \hat{S}_i$ to the prover. The prover sends $\mathsf{agg}_{\mathsf{in}}$ and proves it was constructed correctly using an instance of MIPP. The

$$\begin{array}{lcl}
\underline{\mathsf{HICIAP.Prove}\begin{pmatrix}(\mathsf{ck},\mathsf{crs}),\hat{\mathbf{S}},\\(a_0,\mathbf{A}',\mathbf{B}',\mathbf{C}')\end{pmatrix}} & & \underline{\mathsf{HICIAP.Vfy}((\mathsf{srs},\mathsf{crs}),\mathsf{com}_{\mathsf{in}})} \\
(\mathbf{A}',\mathbf{B}',\mathbf{C}') \leftarrow \mathsf{Groth16.Rerand}^{\mathbb{M}'}(\mathbf{A}',\mathbf{B}',\mathbf{C}') & & \\
z_1,z_2,z_3,z_4 \leftarrow \mathbb{F} & & \\
\mathbf{A} := \mathbf{A}' \parallel [z_1]_1 \parallel [z_2]_1 \in \mathbb{G}_1^n & & \\
\mathbf{B} := \mathbf{B}' \parallel [\gamma]_2 \parallel [\delta]_2 \in \mathbb{G}_2^n & & \\
\mathbf{C} := \mathbf{C}' \parallel [1]_1 \parallel [z_2]_1 \in \mathbb{G}_1^n & & \\
\mathsf{com}_{a_0} := a_0P_1 + z_1P_2 + z_3P_3 & & \\
\mathsf{com}_A := \mathbf{A} * \mathsf{ck}_1 & & \\
\mathsf{com}_B := \mathsf{ck}_2 * \mathbf{B} & & \\
\mathsf{com}_C := e([z_4]_1, \mathsf{ck}_3) \cdot (\mathbf{C} * \mathsf{ck}_1) & \xrightarrow{\mathsf{com}_{a_0},\mathsf{com}_A,\mathsf{com}_B,\mathsf{com}_C} & \\
& \xleftarrow{r} & r \leftarrow \mathbb{F} \\
\mathbf{r} := (r, r^2, \ldots, r^n) & & \\
\mathbf{r}' := \mathbf{r}_{[:n-2]} & & \\
\mathsf{agg}_{\mathsf{in}} := \hat{\mathbf{S}}^{\mathbf{r}'} & & \\
\mathsf{agg}_C := \mathbf{C}^{\mathbf{r}} & & \\
W := \left[z_1 r^{n-1}\right]_1 + \sum_{i=1}^{n-2} r^i a_0 W_0 & \xrightarrow{\mathsf{agg}_{\mathsf{in}},\mathsf{agg}_C,W} & \\
& \xleftrightarrow{\mathsf{MIPP}_k(\mathsf{ck},(\mathsf{com}_{\mathsf{in}},\mathsf{agg}_{\mathsf{in}},\mathbf{r}'),\hat{\mathbf{S}})} & J := e(\mathsf{agg}_{\mathsf{in}}, [\gamma]_2) \\
& \xleftrightarrow{\mathsf{HMIPP}(\mathsf{ck},(\mathsf{com}_C,\mathsf{agg}_C,\mathbf{r}),(\mathbf{C},z_4))} & \\
G_1 := \sum_{i=1}^{n-2} r^i W_0, \quad G_2 := \left[r^{n-1}\right]_1 & \xleftrightarrow{\mathsf{HWW}\begin{pmatrix}(\mathsf{com}_{a_0},W,P_1,P_2,P_3,G_1,G_2),\\(a_0,z_1,z_3)\end{pmatrix}} & G_1 := \sum_{i=1}^{n-2} r^i W_0, \quad G_2 := \left[r^{n-1}\right]_1 \\
\mathsf{agg}_{AB} := \mathbf{A}^{\mathbf{r}} * \mathbf{B} & \xleftrightarrow{\mathsf{TIPP}(\mathsf{ck},(\mathsf{com}_A,\mathsf{com}_B,\mathsf{agg}_{AB},\mathbf{r}),(\mathbf{A},\mathbf{B}))} & \mathsf{agg}_{AB} := \prod_{i=1}^{n} e([\alpha]_1,[\beta]_2)^{r^i} \cdot J \\
\textbf{return } o := (z_1, z_3) & & \quad \cdot e(W,[\gamma]_2) \cdot e(\mathsf{agg}_C,[\delta]_2)
\end{array}$$

Fig. 4: The (interactive) HICIAP protocol. Vfy accepts iff all subprotocols MIPP, HMIPP, HWW, and TIPP accept. The group elements $W_0, [\alpha]_1, [\beta]_2, [\delta]_2, [\gamma]_2$ are supplied by crs. The values P_1, P_2, P_3 used to compute com_{a_0} is a Pedersen commitment basis. The Rerand procedure is only executed on the indices in the (log-sized) masking set $\mathbb{M}' = \mathbb{M} \cup \{n-2\}$, where $\mathbb{M}$ is defined in the proof of Lemma 4. It is the identity function everywhere else.

verifier checks this with respect to $\mathsf{com}_{\mathsf{in}}$, which it can compute from public inputs independently of the proof-specific r values. With these two optimizations, the final verifier equation is

$$\mathsf{agg}_{AB} \overset{?}{=} e([\alpha]_1, [\beta]_2)^{\frac{r^{n+1}-1}{r-1}} \cdot \mathsf{agg}_C \cdot e(W, [\gamma]_2) \cdot e(\mathsf{agg}_{\mathsf{in}}, [\gamma]_2).$$

It is important to reiterate that, while this resembles HICIAP's verification equation,[3] the protocol outlined above is not zero-knowledge. W leaks a_0; agg_{AB}, com_A, and com_B leak parts of $\mathbf{A}$ and $\mathbf{B}$; and agg_C and com_C leak parts of $\mathbf{C}$. In order to achieve zero-knowledge, we blind all of these values using the explicit blinders $z_1, \ldots, z_4 \in \mathbb{F}$ and the implicit blinders in the Groth16.Rerand sub-procedure.

[3]For clarity's sake, however, the Vfy algorithm in Figure 4 does not include the geometric sum formula, or any other arithmetic optimizations.

B. HICIAP details

We now give the formal definitions of the HICIAP relations and procedures. The HICIAP relation for a fixed n, where n is a power of two, is defined to be

$$R_{\mathsf{HICIAP}} = \left\{ \begin{array}{c} \left(\begin{array}{c} \mathsf{ck}, \mathsf{crs}, \mathsf{com}_{\mathsf{in}}, \{\hat{S}\}_{i=1}^{n-2}; \\ a_0, \{\pi_i\}_{i=1}^{n-2} \end{array} \right) : \\ \mathsf{com}_{\mathsf{in}} = \hat{\mathbf{S}} * \mathsf{ck}_1 \\ \bigwedge_{i=1}^{n-2} \mathsf{Groth16.Vfy}(\mathsf{crs}, \pi_i, (a_0, \hat{S}_i)) \end{array} \right\}.$$

The associated protocol is given in Figure 4, and is made non-interactive by applying the Fiat-Shamir transform [FS87].

Note that Prove outputs an opening $o = (z_1, z_3)$ of com_{a_0}. This opening is used for linkage proofs, which show that multiple HICIAP proofs share the same a_0. Formally, this relation is

$$R_{\mathsf{link}} = \left\{ \begin{array}{c} \left(\{\hat{\pi}_i\}_{i=1}^{t}; a_0, \{o_i\}_{i=1}^{t}\right) : \\ \bigwedge_{i=1}^{t} \mathsf{com}_{a_0}^{(i)} = a_0 P_1 + z_1^{(i)} P_2 + z_3^{(i)} P_3 \end{array} \right\},$$

where $\mathsf{com}_{a_0}^{(i)}$ comes from $\hat{\pi}_i$, $(z_1^{(i)}, z_3^{(i)})$ come from o_i, and $P_1, P_2, P_3 \in \mathbb{G}_1$ is a Pedersen basis shared by all HICIAP instances. The LinkProve and LinkVfy algorithms are constructed using the generic Σ-protocol framework defined by Camenisch and Stadler [CS97]. We defer their full description and security analysis to the extended version of this paper [RMM21].

For the last relation, recall from the intuition that the value W in HICIAP proofs must be proven to represent the value a_0 on the wire W_0 and no more (i.e., it must not allow the prover to cancel other wire values). We call this the *hidden wire well-formedness* (HWW) relation:

$$R_{\mathsf{HWW}} := PK \left\{ \begin{array}{c} \left(\begin{array}{c} U, V, \{G_i\}_{i=1}^{5} \in \mathbb{G}_1; \\ w, x, y \in \mathbb{F} \end{array} \right) : \\ U = wG_1 + xG_2 + yG_3 \\ \bigwedge \quad V = wG_4 + xG_5 \end{array} \right\}$$

Like Link, the HWW proof system is a Σ-protocol constructed using the Camenisch-Stadler framework. The protocol is described and proven secure in the extended version of this paper.

We claim that HICIAP is a zkSNARK for the R_{HICIAP} relation. The proofs of the below theorems can be found in Appendix B. Lastly, we note that the requirement that n is a power of 2 greater than or equal to 16 is not a barrier to usage, since proofs (and their prepared public inputs) can be duplicated arbitrarily many times to pad the input of the HICIAP algorithms.

Theorem 2 (HICIAP Soundness)**.** HICIAP *on* $n-2$ *proofs has witness-extended emulation against algebraic adversaries under the* DL, n-ASDBP, *and* $2n$-SDH *assumptions.*

Theorem 3 (HICIAP Perfect Honest Verifier Zero Knowldege)**.** *The* HICIAP *protocol is perfect HVZK, provided that* $n \geq 16$.

VII. Implementation and evaluation

We now detail the design and evaluation of SnarkBlock.

A. Implementation and setup

Hardware. All benchmarks were performed on a desktop computer with a 2021 Intel i9-11900KB CPU with 8 physical cores and 64GiB RAM running Ubuntu 20.04 with kernel 5.11.0-40-generic.

Code. SnarkBlock consists of 4.3k lines of Rust code[4] using the Arkworks [Ar21] zkSNARK crates and Rayon for parallelization where possible. The Criterion-rs crate was used for all benchmarks and statistics.

Statistics. Performance measurements are for medians and include error bars indicating the 95% confidence interval. These are not visible: over all benchmarks, the maximum relative standard error of the median is 1.6%.

Instantiating cryptographic primitives. We set $\lambda = 128$. We use BLS12-381 [Bow17] for our Groth16 and HICIAP proofs, and Jubjub [ZCa19] for Schnorr signatures. We use hash functions $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ for identity registration, tag computation, and Schnorr signatures, respectively. Specifically, the commitment scheme used for R_{isu} is $\mathsf{Com}(m, r) := \mathsf{H}_1(r \| m)$ and the PRF scheme used for R_{chunk} is $\mathsf{Prf}_k(m) := \mathsf{H}_2(k \| m)$. Each H_i is a domain-separated instantiation of the Poseidon family [GKK+19], configured to be compatible with BLS12-381 and a 128-bit security level (i.e., $\alpha = 5$ and capacity $= 1$).

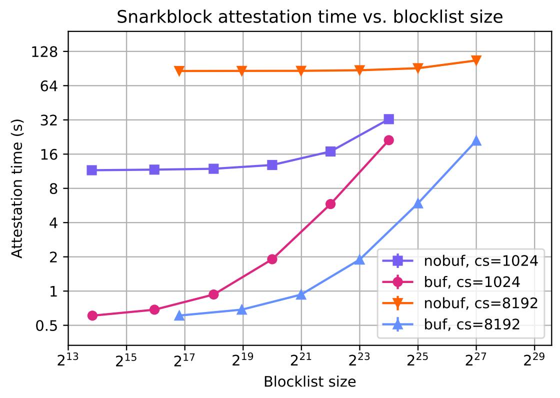
B. Evaluation

Benchmarks are given in Figures 5 and 6. Lines marked *buf* were benchmarked using a size-14 buffer of 16-element chunks. Lines marked *nobuf* used no buffer. The *cs* parameter refers to chunk size.

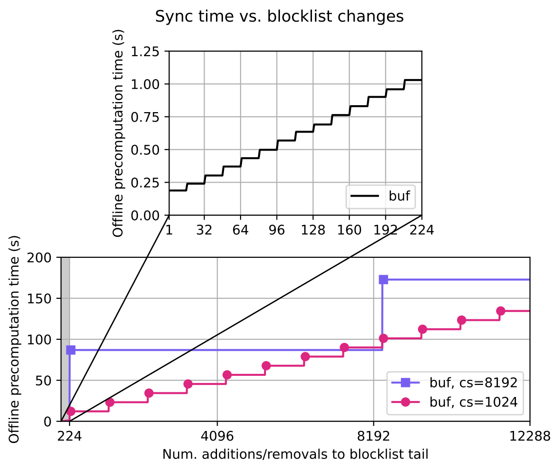
Figure 5a gives the time clients take to attest to non-membership on a blocklist that has recently changed. Specifically, this is the time it takes for a user to re-compute the last Groth16 chunk proof; compute HICIAP proofs over the blocklist, buffer (if the buffer exists), issuance, and tag well-formedness proofs;[5] and compute

[4]The HICIAP crate source code can be found at https://github.com/rozbb/hiciap and the SnarkBlock crate source code can be found at https://github.com/rozbb/snarkblock.

[5]For speed, we combine R_{isu} and R_{tag} into a single circuit in our implementation. Thus there are only 2 proofs to link in an unbuffered SnarkBlock attestation.

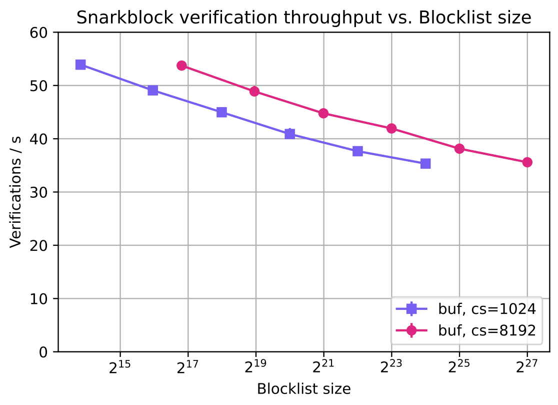


(a) Online costs for proving non-membership vs. blocklist size

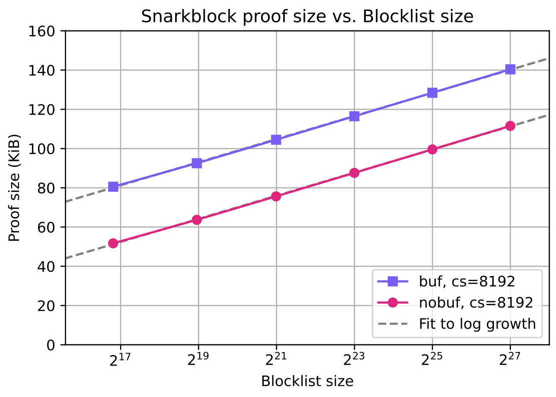


(b) Precomputation cost vs. additions/removals to the blocklist

Fig. 5: Client-side performance for SNARKBLOCK



(a) SNARKBLOCK attestation verifications per second vs. blocklist size.



(b) SNARKBLOCK attestation proof size vs. blocklist size. Varying chunk size only shifts the x-axis.

Fig. 6: Server-side performance for SNARKBLOCK

the link proof over those. Separately, Figure 5b gives the offline computation a client must do as a function of the number of additions/removals to the blocklist (e.g. per day). This includes syncing chunks and precomputing its Groth16 issuance and tag well-formedness proofs for the next attestation.

Figure 6 gives proof sizes and throughput for server verification. These graphs, which are semi-log scale, show that, unlike previous work, SNARKBLOCK proofs scale logarithmically with the number of elements in the blocklist.

VIII. DISCUSSION

We now discuss real-world performance and possible extensions.

A. *Is* SNARKBLOCK *practical?*

Attestation latency. How long can attestation take in practice? A client that computes an attestation in the background while a user drafts their post or comment adds no latency to the user's workflow. When the expected time to write a comment is lower than attestation time (e.g., writing a tweet), then the comment must be queued and posted by the client software when attestation is complete. While this is acceptable in many cases, it renders SNARKBLOCK impractical for real-time chat when blocklists are large.

Operating Costs. SNARKBLOCK can be used when 1) logging in to a pseudonymous session, or 2) posting or commenting anonymously. The latter puts more load on a server. We use it as an estimate for worst-case performance costs.

English language Wikipedia had 2 edits per second

	Client Attestation	Server Verification	Proof Size	
BLAC [TAKS10]	$2nM_{\mathbb{G}_1}$	$(2n+4)M_{\mathbb{G}_1}+2P$	Abs.	$(3n+12)\lvert\mathbb{F}\rvert+(n+3)\lvert\mathbb{G}_1\rvert$
			Real	$528\text{B}+n\cdot 144\text{B}$
SnarkBlock	$(197+10c)M_{\mathbb{G}_1}$ $+(160+10c)M_{\mathbb{G}_2}+2M_{\mathbb{G}_T}$ $+(244+15c)P$	$25M_{\mathbb{G}_1}+38P$ $+(46+10\log_2(c))M_{\mathbb{G}_T}$	Abs.	$8\lvert\mathbb{F}\rvert+29\lvert\mathbb{G}_1\rvert+14\lvert\mathbb{G}_2\rvert$ $+(48+10\log_2 c)\lvert\mathbb{G}_T\rvert$
			Real	$29.3\text{KiB}+\log_2(c)\cdot 5.6\text{KiB}$

Legend: $n=$ Blocklist length, $c=$ Num. chunks, $M_{\mathbb{G}}=$ Var. base MSM in $\mathbb{G}$, $P=$ Pairing op., $\lvert\mathbb{G}\rvert=$ Size of group elem., $\lvert\mathbb{F}\rvert=$ Size of scalar field elem.

TABLE I: BLAC and SnarkBlock operation counts and proof sizes. SnarkBlock operation counts assume a fully synchronized client and an unbuffered blocklist. The top subcell in the Proof Size column represents abstract element counts. The bottom subcell represents the byte count when instantiated with BLS12-381.

in 2021 [Wik] and Reddit had 64 comments per second in 2020 [Red20]. Estimating from event logs for June through October 2020, English language Wikipedia has about 2k bans per day, of which 250 (12.5%) were permanent. Assuming a similar ban rate, Reddit has at least 8k permanent bans per day and perhaps 32k temporary.

An Amazon EC2 c5.4xLarge costs about \$10 USD per day if reserved for a year.[6] For a blocklist of 2^{24} entries, SnarkBlock handles at least 35 attestations per second. At Reddit's scale, deployed in the more resource-intensive attestation-per-comment mode, SnarkBlock costs on the order of \$20 per day when pessimistically assuming full EC2 retail pricing at scale. With generous allowances for CPU differences and virtualization overhead, SnarkBlock is at most \$200 per day in the worst case. For reference, Facebook pays moderators in the US \$120 a day [Sal19].

A final consideration is increased bandwidth usage by the server. A SnarkBlock attestation for a 4M-entry blocklist is 130KiB, at least two orders of magnitude larger than an average text comment it would accompany in the fully anonymous setting. 130KiB, however, is dwarfed by the size of image and video files uploaded to many service providers. Moreover, inbound traffic is typically a small fraction of total traffic for web services. So much so that on EC2, for example, it's free.

B. Client side performance vs BLAC

SnarkBlock's main advantage over BLAC is logarithmic server-side scaling. Nonetheless, we briefly discuss client-side performance. The biggest problem for BLAC, surprisingly, is proof size. A blocklist with 4M bans yields proofs of 549MiB. In contrast, a SnarkBlock attestation is less than 200KiB for a 134 million entry list. On a 50Mbps connection, which is $5\times$ the upstream bandwidth of the median US household [FCC20],[7] uploading a BLAC attestation would take 90s. Even if both have a 100Mbps fiber connection, SnarkBlock can compute and upload the attestation before a BLAC proof would upload.

What if we ignore proof size? Although Tsang et al. give benchmarks for BLAC, they are on 10+ year old hardware using the very dated PBC library [Lyn] for pairings. Luckily, Tsang et al. also characterize their system's performance in terms of group operations. In lieu of a reimplementation, we report these measurements and give the equivalent values for SnarkBlock in Table I.

SnarkBlock pays an initial overhead in terms of upfront costs (e.g., the 244 pairings). The major advantage for SnarkBlock is that its operations are per chunk as opposed to per element. Ignoring constants, SnarkBlock is faster for proving whenever $2nM_{\mathbb{G}_1} > \frac{10nM_{\mathbb{G}_1}}{s}+\frac{10nM_{\mathbb{G}_2}}{s}+\frac{15nP}{s}$, where $s=n/c$ is the chunk size. Thus, as the blocklist size grows, SnarkBlock will outperform as long as $s>5+12.5o$ where $o=\frac{P}{M_{\mathbb{G}_1}}$ is the overhead for pairings relative to $\mathbb{G}_1$ multiplications.[8] On our benchmark system, $\mathbb{G}_2$ multiplications are about 3 times $\mathbb{G}_1$, and pairings twice that. i.e., $o\approx 6$.

Unfortunately, giving a precise estimate for the transition point is impossible with only group operations: we need to compare runtimes to a full reimplementation of BLAC. Real-world performance will differ significantly from group operation counts due to parallelization and other optimizations. Indeed, SnarkBlock outperforms estimates based on group operations and benchmarked operation times.

SnarkBlock has one substantial cost that BLAC does not: SnarkBlock requires periodic sync computations for blocklist additions and removals. Per Figure 5, this is less than 200s for every 12k additions, with appropriate batching or buffering. For much faster churning

[6]With 16 virtual Xeon CPUs and 30GB of memory, this is a decent analog to our test system since in testing, SnarkBlock never exceeded 20 GB of memory for verification.

[7]FCC measurements are a trailing indicator. The latest report, released in Sep. 2020 [FCC20], is for data as of Dec. 2018. For Oct. 2021, Speedtest.net reports its US users have upstream *averages* of 19.18Mbps for wired connections and 8.81Mbps for mobile.

[8]Since $M_{\mathbb{G}_2} < P$, we can approximate them as the same.

lists, e.g., 64k additions per day, BLAC would have a large initial advantage by avoiding these recomputation costs. But at even 8k additions per day, the service provider will exceed 2^{22} bans within two years. At this point, under reasonable bandwidth assumptions, SNARKBLOCK will outperform BLAC.

C. Cold Start

One significant caveat for SNARKBLOCK is that a new user of a system with a pre-existing blocklist must do significant work to sync the entire blocklist and compute the chunk proofs.

One option is to leverage the issuance date for a user's identity and allow them to skip proving membership in blocklist chunks whose last entry is before they joined. This can be done directly now, albeit at the cost of leaking the user's approximate join time for, e.g., a particular forum. Specifically, a given service provider can use a custom CRS for their Groth16 chunk proofs. They can then, using the CRS trapdoor, give each new user non-membership chunk proofs for earlier portions of the blocklist. Crucially, proving with a trapdoor is constant-time, so this process is efficient.

We leave to future work the question of how to build a general trapdoor for cold start. In particular, it should be possible skip chunk proofs whose last entry was inserted before the issuance data of the user's identity.

IX. RELATED WORK

For a full formalization of privacy preserving blocklists, we refer the reader to excellent SoK of Henry and Goldberg [HG11]. This also describes a number of interesting hybrid systems that can be constructed in a black-box way from either SNARKBLOCK or BLAC and allow for pruning of blocklists.

A. Blocklists

The work closest to ours is the ZKBL approach introduced in BLAC [TAKS10]. As discussed in prior sections, by replacing the zero-knowledge proofs in BLAC with our novel proving system HICIAP, we get a system that offers logarithmic verification time and proof size, rather than linear. Further, we extend the system to support federated identities.

Also close to our work is the windowed approach from PEREA [TAKS08], also by the authors of BLAC. In PEREA, users are issued a finite number of one-time-use identity tickets for use during a revocation window, e.g., one month. To complete an action, a user must prove none of those tickets are in the blocklist. A user computes the same proof to get the next set of tickets. Verification time is proportional to size of the revocation window, not the total size of the blocklist. It has a number of drawbacks for broad deployment on the web:

1) Issuing users a small number of tickets is feasible for individual low-volume sites, but the limit would apply to all sites in a federated system.
2) The approach is inherently centralized. All blocklists must be registered with the single identity provider to ensure non-membership before reissuing identities.
3) Service providers must react quickly to ban users, since bans expire once the user gets new identities. The exact time depends on configuration; PEREA gives the example of a 1-hour window for a site like Wikipedia.

Finally, a number of systems provide weaker anonymity. One line of work relies on a trusted third party to revoke anonymity, e.g, [Cha85], [Cv91], [BMW03]. Another approach is to leverage blind signatures to remove the linkage between, e.g., an IP address, and the pseudonym, e.g., [JKTS07], [TKCS11], [LH10]. These schemes only provide pseudonymity, allowing the linking of pseudonymous posts across different platforms. In contrast, SNARKBLOCK provides anonymity and does not trust a third party to safeguard user identities.

B. Zero-knowledge proofs

Our HICIAP protocol consists of a non-membership proof and a proof that a revocation tag has been computed correctly. Bayer and Groth design a non-membership proof [BG13] with logarithmic proof size and no trusted setup, but they have (quasi-)linear prover and verifier costs. Non-membership proofs can also be constructed in groups of unknown order [CL02], [BCFK19], and have constant verifier time and prover time. However, it is not obvious how to apply these techniques to a blocklist without requiring a finite number of tickets per user as in PEREA.

An alternative and thus far unexplored direction for proving blocklist non-membership is recent advances in recursive zero-knowledge proofs using techniques first introduced by Bowe et al. for Halo [BGH19]. Halo-like schemes, formalized in [BCMS20] as *accumulator schemes*, have been extended to a wider variety of polynomial commitment schemes in [BDFG21]. These use Bulletproofs [BBB^+18] as a building block, which introduces a linear verification time component to the constructions. This cost is typically small if the individual computation step is small, but leads to a different set of design tradeoffs than recursive proofs with fully succinct verifiers. Bünz et al. [BCL^+21] and Kothapalli et al. [KST21] improve upon these results.

One key challenge to using Halo-like techniques is the concrete cost of recursion. With SNARKBLOCK, aggregation costs are less than $8\times$ the cost of native verification, keeping online costs low. For Halo like systems, these costs depend heavily on the exact approach taken.

References

[Ar21] Arkworks-rs. *Arkworks Ecosystem Homepage*, 2021. https://arkworks.rs/.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73. Springer, Heidelberg, May 2004.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

[BCD$^+$17] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. Cryptology ePrint Archive, Report 2017/043, 2017. https://eprint.iacr.org/2017/043.

[BCFK19] Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. https://eprint.iacr.org/2019/1255.

[BCG$^+$15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In IEEE S&P 2015 [IEE15], pages 287–304.

[BCL$^+$21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Malkin and Peikert [MP21], pages 681–710.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499, 2020. https://eprint.iacr.org/2020/499.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In Malkin and Peikert [MP21], pages 649–680.

[BG13] Stephanie Bayer and Jens Groth. Zero-knowledge argument for polynomial evaluation with application to blacklists. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 646–663. Springer, Heidelberg, May 2013.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050.

[BKSV20] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-SNARK. Cryptology ePrint Archive, Report 2020/811, 2020. https://eprint.iacr.org/2020/811.

[BMM$^+$20] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177, 2020. https://eprint.iacr.org/2019/1177.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

[Bow17] Sean Bowe. *BLS12-381: New zk-SNARK Elliptic Curve Construction*, 2017. https://electriccoin.co/blog/new-snark-curve/.

[CCDW20] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P. Ward. Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522, 2020. https://ia.cr/2020/1522.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical report, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, April 1991.

[Fac] Facebook. Help Center - What names are allowed on Facebook? https://www.facebook.com/help/112146705538576. Accessed on Nov. 26, 2021.

[FCC20] FCC. Internet access services: Status as of december 31, 2018. https://docs.fcc.gov/public/attachments/DOC-366980A1.pdf, 2020.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[Fuc17] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. Cryptology ePrint Archive, Report 2017/587, 2017. https://eprint.iacr.org/2017/587.

[GKK$^+$19] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. https://eprint.iacr.org/2019/458.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[HG11] Ryan Henry and Ian Goldberg. Formalizing anonymous blacklisting systems. In *2011 IEEE Symposium on Security and Privacy*, pages 81–95. IEEE Computer Society Press, May 2011.

[HKR19] Max Hoffmann, Michael Klooß, and Andy Rupp. Efficient zero-knowledge arguments in the discrete log setting, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2093–2110. ACM Press, November 2019.

[IEE15] *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015.

[JKTS07] Peter C. Johnson, Apu Kapadia, Patrick P. Tsang, and Sean W. Smith. Nymble: Anonymous IP-address blocking. In Nikita Borisov and Philippe Golle, editors, *PET 2007*, volume 4776 of *LNCS*, pages 113–133. Springer, Heidelberg, June 2007.

[KST21] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Report 2021/370, 2021. https://eprint.iacr.org/2021/370.

[LH10] Zi Lin and Nicholas Hopper. Jack: Scalable accumulator-based nymble system. pages 53–62, 01 2010.

[LLX07] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 253–269. Springer, Heidelberg, June 2007.

[Lyn] Ben Lynn. PBC Library - Pairing-Based Cryptography - About.

[MP21] Tal Malkin and Chris Peikert, editors. *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, Virtual Event, August 2021. Springer, Heidelberg.

[Rad21] Radiolab. *The Ceremony*, 2021. https://www.wnycstudios.org/podcasts/radiolab/articles/ceremony.

[Red20] Reddit Staff. Reddit in 2020. (https://old.reddit.com/r/blog/comments/k967mm/reddit_in_2020/, 2020.

[RMM21] Michael Rosenberg, Mary Maller, and Ian Miers. Snarkblock: Federated anonymous blocklisting from hidden common input aggregate proofs. Cryptology ePrint Archive, Report 2021/1577, 2021. https://ia.cr/2021/1577.

[Sal19] Sara Salinas. Facebook pays the reviewers filtering porn and murder a tiny fraction of its median salary, explosive report says. https://www.cnbc.com/2019/02/25/facebook-pays-content-moderators-a-fraction-of-median-salary-report.html, Feb 2019.

[TAKS08] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. PEREA: towards practical TTP-free revocation in anonymous authentication. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 333–344. ACM Press, October 2008.

[TAKS10] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. BLAC: Revoking Repeatedly Misbehaving Anonymous Users without Relying on TTPs. *ACM Transactions on Information and System Security*, 13(4):1–33, December 2010.

[TKCS11] P. P. Tsang, A. Kapadia, C. Cornelius, and S. W. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Transactions on Dependable and Secure Computing*, 8(2):256–269, 2011.

[VB20] Giuseppe Vitto and Alex Biryukov. Dynamic universal accumulator with batch update over bilinear groups. Cryptology ePrint Archive, Report 2020/777, 2020. https://eprint.iacr.org/2020/777.

[Wik] https://stats.wikimedia.org/#/en.wikipedia.org/contributing/edits/normal|bar|2020-11-04~2021-11-24|~total|monthly.

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018.

[ZCa19] ZCash. *What is Jubjub?*, 2019. https://z.cash/technology/jubjub/.

Appendix

A. Cryptographic Definitions

We leave the full definitions of the cryptographic assumptions we rely on for the extended version of this paper [RMM21]. We will use the definition of the Discrete Log (DL) assumption from [FKL18] and the definitions of the q-ASDBP and q-SDH assumptions from [BMM+20].

We will use the definitions of *computation knowledge soundness* and *perfect honest-verifier zero-knowledge* (HVZK) from [BMM+20] in the following proofs of soundness and zero-knowledge.

B. Deferred proofs

Recall R_{HMIPP} is a zero-knowledge ("hiding") version of the $R_{\mathsf{MIPP}-k}$ relation. Both relations are detailed in Figure 1, and their proof systems are given in [BMM+20]. We defer the proof of knowledge soundness to the extended version of this paper [RMM21].

Lemma 4. *The* HMIPP *protocol is perfect HVZK. (Lemma 5 in [BMM+20])*

Lemma 5 (HMIPP Computational Knowledge Soundness)**.** HMIPP *on n elements is computationally knowledge sound against algebraic adversaries under the n-*ASDBP *and $2n$-*SDH *assumptions.*

Recall R_{HWW} is the proof of hidden wire well-formedness described in Section VI. We state its theorems of soundness and zero-knowledge below, and defer the proofs and full description of the algorithms to the extended version of this paper.

Lemma 6. HWW *is Perfect HVZK.*

Lemma 7. HWW *is statistically knowledge-sound.*

Theorem 1 (SnarkBlock Security)**.** SnarkBlock *described in Figure 3 is blocklistable, anonymous and non-frameable provided that Groth16 and* HICIAP *proofs are knowledge-sound and subversion zero-knowledge; Schnorr signatures are unforgeable;* Prf *is pseudorandom; and* Com *is binding and hiding.*

Proof Sketch. **Blocklistability.** Let A be an adversary that breaks blocklistability. Then A generates a verifying attestation $(\pi_{\mathsf{zkbl}}, \mathsf{tag}, \mathsf{nonce})$. Either an extractor can output k such that $\mathsf{tag} = \mathsf{Prf}_k(n)$; or $\hat{\pi}_{\mathsf{tag}}$ is a forgery for HICIAP and we cannot extract a verifying Groth16 statement and proof k, π_{tag} breaking knowledge soundness; or π_{tag} is a forgery for Groth16 and $\mathsf{tag} \neq \mathsf{Prf}_k(\mathsf{nonce})$ breaking knowledge soundness.

Either an extractor can output a verifying signature σ under some identity providers public key pk_{i^*} on the message $\mathsf{com} = \mathsf{Com}(k, r)$ the same k and some r; or $\hat{\pi}_{\mathsf{link}}$ is a forgery for Link breaking knowledge soundness; or the adversary can find $\mathsf{com} = \mathsf{Com}(k', r')$ for different k', r' breaking binding; or $\hat{\pi}_{\mathsf{isu}}$ is a forgery for HICIAP and we cannot extract a verifying Groth16 proof π_{isu} for k breaking knowledge soundness; or $((k, \mathscr{I}), \pi_{\mathsf{isu}})$ is a forgery for Groth16 breaking knowledge soundness. If σ is a verifying signature then either pk_{i^*} authenticated com at some point, or we break unforgeability of the signature scheme.

If σ has been authenticated by pk_{i^*} that gets blocked then $(\mathsf{tag}, \mathsf{nonce})$ gets added to $\mathscr{L}$. If A later generates a verifying attestation with respect to the same σ then either $\hat{\pi}_{\mathsf{link}}$ is a forgery for Link breaking knowledge soundness; or the adversary can find $\mathsf{com} = \mathsf{Com}(k', r')$ for different k', r' breaking binding; or $\hat{\pi}_{\mathsf{chunk}}$ is a forgery for HICIAP and we cannot extract verifying Groth16 proofs π_{chunk_j} for k breaking knowledge soundness; or π_{chunk_j} is a forgery for Groth16 for some j and $\mathsf{Prf}_k(\mathsf{nonce}^*) = \mathsf{tag}^*$ for some $(\mathsf{nonce}^*, \mathsf{tag}^*) \in \mathscr{L}$ breaking knowledge soundness; or σ is never associated with a blocked session and A does not break blocklistability.

Non-Frameability. If an adversarial identity provider prevents an honest user from authenticating then they must get some $(\mathsf{nonce}, \mathsf{tag})$ added to $\mathscr{L}$ such that $\mathsf{tag} = \mathsf{Prf}_k(\mathsf{nonce})$ for an honest user's k. By the pseudorandomness of Prf and the anonymity of SNARKBLOCK, the probability that they guess any such tag is negligible.

Anonymity. We claim that the transcript between an honest user and any number of identity providers and service providers is uncorrelated. By the hiding of Com we have that com reveals no information about k (and uses distinct r each registration). By the zero-knowledge of HICIAP we have that π_{zkbl} reveals no information (even to the identity providers). nonce is chosen uniformly at random for each session associated with k. By the pseudorandomness of Prf, tag is indistinguishable from random for users that don't know k and thus reveals no information about k. Thus the scheme is anonymous. □

Theorem 2 (HICIAP Soundness). *HICIAP on $n-2$ proofs has witness-extended emulation against algebraic adversaries under the* DL, n-ASDBP, *and* $2n$-SDH *assumptions.*

Proof. We wish to show that, there exists an expected polynomial time HICIAP extractor $\mathsf{E}^{\mathsf{P}^*}_{\mathsf{HICIAP}}(\mathsf{ck}, \mathsf{crs}, \hat{\mathbf{S}})$ which outputs a witness $(a_0, \mathbf{A}', \mathbf{B}', \mathbf{C}')$ such that for all $i = 1, \ldots, n-2$,

$$\mathsf{Groth16.Vfy}\left(\mathsf{crs}, (A'_i, B'_i, C'_i), (a_0, \hat{S}_i)\right).$$

By Theorem 3 of [BMM+20], there is an expected polynomial time extractor $\mathsf{E}_{\mathsf{TIPP}}$ for $\mathsf{TIPP}(\mathsf{com}_A, \mathsf{com}_B, \mathsf{agg}_{AB}, \mathbf{r})$ which extracts $(\mathbf{A}, \mathbf{B})$ such that $\mathsf{com}_A = \mathbf{A} * \mathsf{ck}_1$, $\mathsf{com}_B = \mathsf{ck}_2 * \mathbf{B}$, and $\mathsf{agg}_{AB} = \mathbf{A}^{\mathbf{r}} * \mathbf{B}$. By Theorem 6 of [BMM+20], there is an expected polynomial time extractor $\mathsf{E}_{\mathsf{MIPP}-k}$ for $\mathsf{MIPP}_k(\mathsf{com}_{\mathsf{in}}, \mathsf{agg}_{\mathsf{in}}, \mathbf{r}')$ which extracts $\hat{\mathbf{S}}'$ such that $\hat{\mathbf{S}}' * \mathsf{ck}_{1,[:n-2]} = \mathsf{com}_{\mathsf{in}}$ and $(\hat{\mathbf{S}}')^{\mathbf{r}'} = \mathsf{agg}_{\mathsf{in}}$. By Lemma 5, there is an expected polynomial time extractor $\mathsf{E}_{\mathsf{HMIPP}}$ for $\mathsf{HMIPP}(\mathsf{com}_C, \mathsf{agg}_C, \mathbf{r})$ which extracts $(\mathbf{C}, z_4)$ such that $\mathbf{C}^{\mathbf{r}} = \mathsf{agg}_C$ and $\mathsf{com}_C = e(z_4 G, \mathsf{ck}_3) + (\mathbf{C} * \mathsf{ck}_1)$. By Lemma 7, there is an expected polynomial time extractor $\mathsf{E}_{\mathsf{HWW}}(\mathsf{com}_{a_0}, W, P_1, P_2, P_3, G_1, G_2)$ for HWW which extracts (a_0, z_1, z_3) such that $\mathsf{com}_{a_0} = a_0 P_1 + z_1 P_2 + z_3 P_3$ and $W = a_0 G_1 + z_1 G_2$.

Let P^* be a probabilistic prover with fixed randomness and unknown probability ε of producing an argument that accepts. We define an extractor $\mathsf{E}^{\mathsf{P}^*}_{\mathsf{HICIAP}}(\mathsf{ck}, \mathsf{crs}, \hat{\mathbf{S}})$, which extracts the witness $(a_0, \mathbf{A}, \mathbf{B}, \mathbf{C})$, as follows.

First, run HICIAP with a random $r \leftarrow \mathbb{F}$, and run all the subprotocols honestly. Note that, by the definition of witness-extended emulation, if P^* does not produce an accepting transcript tr on the first run, the extractor is allowed to exit early with $(\mathsf{tr}, \perp)$. If tr is accepting, rewind to the point after r is chosen and run $\mathsf{E}_{\mathsf{MIPP}-k}$, $\mathsf{E}_{\mathsf{HMIPP}}$, $\mathsf{E}_{\mathsf{HWW}}$, and $\mathsf{E}_{\mathsf{TIPP}}$ to extract $(a_0, z_1, z_3, z_4, \mathbf{A}, \mathbf{B}, \mathbf{C}, \hat{\mathbf{S}}')$. Finally, output $(a_0, \mathbf{A}_{[:n]}, \mathbf{B}_{[:n]}, \mathbf{C}_{[:n]})$.

Note that $\mathsf{E}_{\mathsf{HICIAP}}$ algorithm is expected polynomial time, since its runtime is at most the sum of the runtimes of $\mathsf{E}_{\mathsf{HMIPP}}$, $\mathsf{E}_{\mathsf{HWW}}$, and $\mathsf{E}_{\mathsf{TIPP}}$, which are assumed to be expected polynomial time.

To prove the claimed relations hold, first note that the commitment $\mathsf{com}_{\mathsf{in}} = \hat{\mathbf{S}} * \mathsf{ck}_{1,[:n-2]}$ is computationally binding under the $(n-2)$-ASDBP assumption, and so, with overwhelming probability, $\hat{\mathbf{S}}' = \hat{\mathbf{S}}$.

It remains to show that, with overwhelming probability, the extracted witness satisfies the Groth16 verification condition. That is, for all $i = 1, \ldots, n$,

$$e(A_i, B_i) = e([\alpha]_1, [\beta]_2) \cdot e(C_i, [\delta]_2) \cdot e\left(\hat{S}_i, [\gamma]_2\right).$$

The commitments $\mathsf{com}_A, \mathsf{com}_B, \mathsf{com}_C$ are computationally binding under the n-ASDBP assumptions. Further, since P_1, P_2, P_3 are unrelated by assumption, the Pedersen commitment com_{a_0} is computationally binding by the DL hardness assumption. Thus, with overwhelming probability, the formal product being evaluated in TIPP is the one committed to by $\mathsf{com}_A, \mathsf{com}_B, \mathsf{com}_C$, i.e.,

$$\begin{aligned}\prod_{i=1}^{n} & e(A_i, B_i)^{x_i} \\ &= \prod_{i=1}^{n-2} e(a_0 W_0, [\gamma]_2)^{x_i} + \prod_{i=1}^{n} \cdot e([\alpha]_1, [\beta]_2)^{x_i} \\ &\quad + \prod_{i=1}^{n-2} \cdot e\left(\hat{S}_i, [\gamma]_2\right)^{x_i} + \prod_{i=1}^{n} e(C_i, [\delta]_2)^{x_i}. \\ &\quad + e([z_1]_1, [\gamma]_2)^{x^{n-1}}\end{aligned}$$

Then by the Schwartz-Zippel lemma, the above relation holds with probability at least $1 - n/p$. Since the above equality directly implies the Groth16 verification condition, the theorem is proved. □

Theorem 3 (HICIAP Perfect Honest Verifier Zero Knowldege). *The HICIAP protocol is perfect HVZK, provided that $n \geq 16$.*

Proof. A HICIAP proof consists of the values

$$\mathsf{com}_{\mathsf{in}}, \mathsf{com}_{a_0}, \mathsf{com}_A, \mathsf{com}_B, \mathsf{com}_C, \mathsf{agg}_{\mathsf{in}}, \mathsf{agg}_C, W,$$
$$\mathsf{tr}_{\mathsf{MIPP}\text{-}k}, \mathsf{tr}_{\mathsf{HMIPP}}, \mathsf{tr}_{\mathsf{HWW}}, \mathsf{tr}_{\mathsf{TIPP}}.$$

We construct a simulator that knows a Groth16 simulation trapdoor τ to crs and which can choose the verifier's randomness in advance, such that the simulated transcript is indistinguishable from an honest prover's transcript. The simulator will also use the simulators described in Lemmas 4 and 6 which generate transcripts for subprotocols HMIPP and HWW,

$$\mathsf{Sim}_{\mathsf{HMIPP}}(\mathsf{ck}, \mathsf{com}_C, \mathsf{agg}_c, \mathbf{r}) \to \mathsf{tr}_{\mathsf{HMIPP}}$$
$$\mathsf{Sim}_{\mathsf{HWW}}(\mathsf{com}_{a_0}, W, P_1, P_2, P_3, G_1, G_2) \to \mathsf{tr}_{\mathsf{HWW}}.$$

The simulator is given the Groth16 prepared public inputs $\hat{\mathbf{S}}$ and behaves as follows.

1) The simulator computes the first prover message $\mathsf{com}_{a_0}, \mathsf{com}_A, \mathsf{com}_B, \mathsf{com}_C$. It chooses randomness $a_0, z_1, z_2 \leftarrow \mathbb{F}$ and $\mathsf{com}_{a_0} \leftarrow \mathbb{G}_1$ and $\mathsf{com}_C \leftarrow \mathbb{G}_T$. It runs
$$(A_i', B_i', C_i') = \mathsf{Sim}_{\mathsf{Groth16}}(\mathsf{crs}, \tau, (a_0, \hat{S}_i))$$
for $1 \le i \le n-2$. It sets
$$\begin{aligned} \mathbf{A} &:= \mathbf{A}' \,||\, [z_1]_1 \,||\, [z_2]_1 \\ \mathbf{B} &:= \mathbf{B}' \,||\, [\gamma]_2 \,||\, [\delta]_2 \\ \mathbf{C} &:= \mathbf{C}' \,||\, [1]_1 \,||\, [z_2]_1 \\ \mathsf{com}_A &:= \mathbf{A} * \mathsf{ck}_1 \\ \mathsf{com}_B &:= \mathsf{ck}_2 * \mathbf{B} \end{aligned}$$
2) The simulator computes the first verifier message honestly and chooses $r \leftarrow \mathbb{F}$ randomly.
3) The simulator uses $\hat{\mathbf{S}}$ to construct $\mathsf{agg}_{\mathsf{in}}$ and $\mathsf{tr}_{\mathsf{MIPP}\text{-}k}$ honestly.
4) The simulator computes the second prover message (agg_C, W) honestly as
$$\mathsf{agg}_C := \mathbf{C}^{\mathbf{r}}$$
$$W := [z_1 r^{n-1}]_1 + \sum_{i=1}^{n-2} r^i a_0 W_0$$
for $\mathbf{r} = (r, r^2, \ldots, r^n)$.
5) The simulator generates a transcript $\mathsf{tr}_{\mathsf{HMIPP}}$ for the HMIPP protocol by running
$$\mathsf{tr}_{\mathsf{HMIPP}} := \mathsf{Sim}_{\mathsf{HMIPP}}(\mathsf{crs}, \mathsf{com}_C, \mathsf{agg}_c, \mathbf{r})$$
6) The simulator generates a transcript $\mathsf{tr}_{\mathsf{HWW}}$ for the HWW protocol by running
$$\mathsf{tr}_{\mathsf{HWW}} := \mathsf{Sim}_{\mathsf{HWW}}(\mathsf{com}_{a_0}, W, P_1, P_2, P_3, \sum_{i=1}^{n-2} r^i W_0, [r^{n-1}]_1)$$
7) The simulator computes agg_{AB} honestly and generates a transcript $\mathsf{tr}_{\mathsf{TIPP}}$ by running
$$\mathsf{tr}_{\mathsf{TIPP}} := \mathsf{TIPP}(\mathsf{ck}, (\mathsf{com}_A, \mathsf{com}_B, \mathsf{agg}_{AB}, \mathbf{r}), \mathbf{A}, \mathbf{B}).$$

We will show that the simulator's transcript is indistinguishable from an honest prover's. We look at the distribution of each of the proof components.

The MIPP **optimization.** We first note that this optimization is simulated perfectly, since it involves no witness values. Specifically, $\mathsf{com}_{\mathsf{in}}$, $\mathsf{agg}_{\mathsf{in}}$, and $\mathsf{tr}_{\mathsf{MIPP}\text{-}k}$ are simulated perfectly, since both the prover and simulator have access to the Groth16 public prepared inputs $\hat{\mathbf{S}}$.

The first prover message. We look at com_{a_0}, com_A, com_B, and com_C. In the real prover execution: com_{a_0} is distributed uniformly at random because it is randomized by z_3; com_A is distributed uniformly at random because it is randomized by z_2; com_B is distributed uniformly at random because it is randomized by B'_{n-2}; com_C is distributed uniformly at random because it is randomized by z_4. In the simulated execution: com_{a_0} is chosen uniformly at random; com_A is distributed uniformly at random because it is randomized by z_2; com_B is distributed uniformly at random because it is randomized by B'_{n-2}; com_C is chosen uniformly at random. Thus both the provers and the simulators first messages are distributed randomly and are indistinguishable.

The second prover message. We second look at agg_C, W. In the real prover execution: agg_C is distributed uniformly at random because it is randomized by C'_{n-2}; W is distributed uniformly at random because it is randomized by z_1. In the simulated execution: com_{a_0} is chosen uniformly at random; agg_C is distributed uniformly at random because it is randomized by C'_{n-2}; com_W is distributed uniformly at random because it is randomized by z_1. Thus both the provers and the simulators second messages are distributed randomly and are indistinguishable.

The hidden MSM argument. We see that $\mathsf{tr}_{\mathsf{HMIPP}}$ generated by the prover and simulator are indistinguishable by the zero-knowledge of HMIPP (Lemma 4).

The HWW argument. We see that $\mathsf{tr}_{\mathsf{HWW}}$ generated by the prover and simulator are indistinguishable by the perfect zero-knowledge of HWW (Lemma 6).

The TIPP argument. In order to argue that $\mathsf{tr}_{\mathsf{TIPP}}$ generated by the prover and simulator are indistinguishable we must look at the rerandomizations of each (A_i, B_i, C_i). The bulk of the following argument consists of demonstrating that enough values in the HICIAP protocol are independent and uniformly distributed. To do this, we associate each iid uniform blinding factor to at most one transcript variable. One thing to be careful about here is enforcing the "at most one" requirement.

Following [HKR19] we define a masking set $\mathbb{M}$ of size $O(\log_2 n)$ that defines a position of randomized values that will ensure the transcripts appear random in the recursion. We track the parts of the TIPP transcript which are functions of $\mathbf{A}, \mathbf{B}, \mathsf{ck}_1, \mathsf{ck}_2$ (where we let $\mathbf{A}$ represent $\mathbf{r} \odot \mathbf{A}$ and ck_1 represent $\mathbf{r}^{-1} \odot \mathsf{ck}_1$ for simplicity). In each round of the TIPP protocol (of which there are $\log n$), the prover sends six values:

$$\begin{aligned} \mathsf{com}_{LA} &:= \mathbf{A}_{[:h]} * \mathsf{ck}_1 & \mathsf{com}_{RA} &:= \mathbf{A}_{[h:]} * \mathsf{ck}_1 \\ \mathsf{com}_{LB} &:= \mathsf{ck}_2 * \mathbf{B}_{[:h]} & \mathsf{com}_{RB} &:= \mathsf{ck}_2 * \mathbf{B}_{[h:]} \\ \mathsf{agg}_{LR} &:= \mathbf{A}_{[:h]} * \mathbf{B}_{[h:]} & \mathsf{agg}_{RL} &:= \mathbf{A}_{[h:]} * \mathbf{B}_{[:h]} \end{aligned}$$

The verifier sends a challenge x, which defines the prover's values for the next round:

$$\begin{aligned} \mathbf{A}' &:= \mathbf{A}_{[:h]} + x \cdot \mathbf{A}_{[h:]} & \mathsf{ck}_1' &:= \mathsf{ck}_{1,[:h]} + x^{-1} \cdot \mathsf{ck}_{1,[h:]} \\ \mathbf{B}' &:= \mathbf{B}_{[:h]} + x^{-1} \cdot \mathbf{B}_{[h:]} & \mathsf{ck}_2' &:= \mathsf{ck}_{2,[:h]} + x \cdot \mathsf{ck}_{2,[h:]} \end{aligned}$$

Note that a randomized A_i value in round k will yield a uniform value of A_j in round $k+1$, where $j \equiv i \pmod{2^{k-1}}$, and similarly for B_i.

With 6 proof elements in each round, we need to ensure there are at least 6 randomizers per round, and that one unique randomizer appears in each proof element. We divide them as 3 randomizers in $\mathbf{A}$ (to randomize $\mathsf{com}_{LA}, \mathsf{com}_{RA}, \mathsf{agg}_{LR}$) and 3 in $\mathbf{B}$ (to randomize $\mathsf{com}_{LB}, \mathsf{com}_{RB}, \mathsf{agg}_{RL}$). We define the masking set

$$\mathbb{M} = \{2^k, 2^k+1\}_{k=2}^{\ell-1} \cup \{2^k - 1\}_{k=2}^{\ell-1},$$

where $\ell = \log_2(n)$. The two sets making up $\mathbb{M}$ are non-overlapping. Note that because $\log_2(n) \geq 4$ we have that $\mathbb{M}$ also does not overlap with the blinders B'_{n-2} or C'_{n-2}.

For the components agg_{LR} and agg_{RL} in the TIPP argument, we must use the fact that with overwhelming probability, none of the components of a Groth16 proof (A_i, B_i, C_i) equals 0. This implies that the rerandomization is a uniform proof of the same statement, and also contains no zeros.

With this in mind we argue that $\mathbb{M}$ is sufficient to randomize the distribution of the com_{LA}, com_{RA}, agg_{LR} components of TIPP. To see this, observe that in round k with verifier challenges $x_0, \ldots, x_{k-1}$

$$\mathsf{com}_{LA} = \prod_{\mathbf{b} \in \{0,1\}^{\ell-k-1}} e\left(\sum_{\mathbf{s} \in \{0,1\}^k} A_{(\mathbf{s},0,\mathbf{b})} f_{k,\mathbf{x}}(\mathbf{s}), \sum_{\mathbf{s} \in \{0,1\}^k} \mathsf{ck}_{A,(\mathbf{s},1,\mathbf{b})} f_{k,\mathbf{x}^{-1}}(\mathbf{s}) \right),$$

$$\mathsf{com}_{RA} = \sum_{\mathbf{b} \in \{0,1\}^{\ell-k-1}} e\left(\sum_{\mathbf{s} \in \{0,1\}^k} A_{(\mathbf{s},1,\mathbf{b})} f_{k,\mathbf{x}}(\mathbf{s}), \sum_{\mathbf{s} \in \{0,1\}^k} \mathsf{ck}_{A,(\mathbf{s},0,\mathbf{b})} f_{k,\mathbf{x}^{-1}}(\mathbf{s}) \right),$$

$$\mathsf{agg}_{LR} = \prod_{\mathbf{b} \in \{0,1\}^{\ell-k-1}} e\left(\sum_{\mathbf{s} \in \{0,1\}^k} A_{(\mathbf{s},0,\mathbf{b})} f_{k,\mathbf{x}}(\mathbf{s}), \sum_{\mathbf{s} \in \{0,1\}^k} B_{(\mathbf{s},1,\mathbf{b})} f_{k,\mathbf{x}^{-1}}(\mathbf{s}) \right),$$

where $f_{k,\mathbf{x}}(\mathbf{s}) := \prod_{j=0}^{k-1} \left(s_j x_{k-j-1} + 1 - s_j \right)$ for $\mathbf{s} \in \{0,1\}^k$ and $A_{\mathbf{b}}$ represents A_i when $\mathbf{b}$ is the binary representation of the integer i. Thus

- $A_{\mathbf{0},0,1,\mathbf{0},1}$ is included in com_{LA} in the kth round and corresponds to the blinder $2^{\ell-k-2}+1$
- $A_{\mathbf{0},1,\mathbf{0}}$ is included in com_{RA} in the kth round and corresponds to the blinder $2^{\ell-k-1}$.
- $A_{\mathbf{0},0,\mathbf{1}}$ is included in agg_{LR} in the kth round and corresponds to the blinder $2^{\ell-k-1}-1$.

Denote

$$R_{LA,k,\boldsymbol{b}} = \sum_{\mathbf{s} \in \{0,1\}^k} \mathsf{ck}_{A,(\mathbf{s},1,\mathbf{b})} f_{k,\mathbf{x}^{-1}}(\mathbf{s})$$

and

$$R_{RA,k,\boldsymbol{b}} = \sum_{\mathbf{s} \in \{0,1\}^k} \mathsf{ck}_{A,(\mathbf{s},0,\mathbf{b})} f_{k,\mathbf{x}^{-1}}(\mathbf{s})$$

and

$$R_{LR,k,\boldsymbol{b}} = \sum_{\mathbf{s} \in \{0,1\}^k} B_{\mathbf{s},1,\mathbf{b}} f_{k,\mathbf{x}^{-1}}(\mathbf{s})$$

Observe that with overwhelming probability, each of $\{R_{LR,k-1,(\mathbf{0},0,\mathbf{1})}\}_{k=1}^{\ell-1}$ are non-zero, depend non-trivially on x_{k-1}, and have no dependence on x_k Thus the $R_{LR,0,(0,\mathbf{0},1)}, \ldots, R_{LR,\ell-2,(0,\mathbf{0},1)}$ are pairwise independent and ensure that: the $A_{0,0,\mathbf{1}}$ term in $\mathsf{agg}_{LR,2}$ (denoting the second round's agg_{LR}) is randomized by x_1 and thus is independent from the $A_{0,0,\mathbf{1}}$ terms in $\mathsf{agg}_{LR,1}$; the $A_{0,0,0,\mathbf{1}}$ term in $\mathsf{agg}_{LR,3}$ is randomized by x_2 and thus is independent from the $A_{0,0,0,\mathbf{1}}$ terms in $\mathsf{agg}_{LR,1}$ and $\mathsf{agg}_{LR,2}$; etc. Thus $A_{\mathbf{0},0,\mathbf{1}}$ perfectly blinds agg_{LR} except with negligible probability.

By the same argument, the $R_{LA,k,(\mathbf{0},0,1,\mathbf{0},1)}$ terms are pairwise independent and the $R_{LB,k,(\mathbf{0},1,\mathbf{0})}$ terms are pairwise independent ensuring independence between the $A_{\mathbf{0},0,1,\mathbf{0},1}$ terms in $\mathsf{com}_{LA,k}$ and $A_{\mathbf{0},1,\mathbf{0}}$ terms in $\mathsf{com}_{LB,k}$ respectively. Thus $A_{\mathbf{0},0,1,\mathbf{0},1}$, $A_{\mathbf{0},1,\mathbf{0}}$ terms perfectly blind com_{LA}, com_{RA} except with negligible probability.

By a symmetric argument we see that $B_{\mathbf{0},0,1,\mathbf{0},1}$, $B_{\mathbf{0},1,\mathbf{0}}$, $B_{\mathbf{0},0,\mathbf{1}}$ perfectly blinds com_{LB}, com_{RB}, agg_{RL} except with negligible probability.

We now consider the penultimate round (round $\ell-1$). Here the proof elements com_{LA}, com_{LB}, com_{RA} and com_{RB} of both the honest prover and simulator take the form

$$\begin{aligned} \mathsf{com}_{LA} = e(A_{\ell-1,1}, H_2), \mathsf{com}_{RA} = e(A_{\ell-1,2}, H_1), \\ \mathsf{com}_{LB} = e(G_1, B_{\ell-1,2}), \mathsf{com}_{RB} = e(G_2, A_{\ell-1,1}) \end{aligned}$$

for $\mathsf{ck}_{1,\ell-1} = (H_1, H_2)$ and $\mathsf{ck}_{2,\ell-1} = (G_1, G_2)$. Thus the proof elements

$$\mathsf{agg}_{LR} = e(A_{\ell-1,1}, B_{\ell-1,2}), \mathsf{agg}_{RL} = e(A_{\ell-1,2}, B_{\ell-1,1})$$

are uniquely determined given com_{LA}, com_{LB}, com_{RA} and com_{RB}. Hence they are sampled from the same distribution. $\square$