Zero-knowledge Authenticator for Blockchain: Policy-private and Obliviously Updateable

Kostas Kryptos Chalkias[1], Deepak Maram[1], Arnab Roy[1], Joy Wang[1], and Aayush Yadav*[2]

[1]Mysten Labs, Palo Alto, CA
[2]George Mason University, Fairfax, VA

May 22, 2025

Abstract

Transaction details and participant identities on the blockchain are often publicly exposed. In this work, we posit that blockchain's transparency should not come at the cost of privacy. To that end, we introduce *zero-knowledge authenticators* (zkAt), a new cryptographic primitive for privacy-preserving authentication on public blockchains. zkAt utilizes zero-knowledge proofs to enable users to authenticate transactions, while keeping the underlying authentication policies private.

Prior solutions for such *policy-private authentication* required the use of threshold signatures, which can only hide the threshold access structure itself. In comparison, zkAt provides privacy for *arbitrarily complex* authentication policies, and offers a richer interface even within the threshold access structure by, for instance, allowing for the combination of signatures under distinct signature schemes.

In order to construct zkAt, we design a compiler that transforms the popular Groth16 non-interactive zero knowledge (NIZK) proof system into a NIZK with equivocable verification keys, a property that we define in this work. Then, for any zkAt constructed using proof systems with this new property, we show that all public information must be independent of the policy, thereby achieving policy-privacy.

Next, we give an extension of zkAt, called zkAt$^+$ wherein, assuming a trusted authority, policies can be updated obliviously in the sense that a third-party learns no new information when a policy is updated by the policy issuer. We also give a theoretical construction for zkAt$^+$ using recursive NIZKs, and explore the integration of zkAt into modern blockchains. Finally, to evaluate their feasibility, we implement both our schemes for a specific threshold access structure. Our findings show that zkAt achieves comparable performance to traditional threshold signatures, while also attaining privacy for significantly more complex policies with very little overhead.

1 Introduction

Blockchain technology, with its decentralized architecture, offers a transparent and immutable ledger of transactions, fostering trust among participants without the need for centralized intermediaries. To authenticate a blockchain transaction, a user creates a digital signature on the transaction. If said signature verifies under the user's signature verification key, then the transaction is permanently added to the blockchain by the validators.

This transparency, however, often comes at the cost of privacy, as transaction details, participant identities, and contract logic are exposed to public scrutiny on most blockchains. While there has been significant progress in enabling confidential transactions with privacy-preserving blockchains, such as Zcash and Monero, and mixing services [GM17, DEFM19, DGL$^+$22, HAB$^+$17, GMM$^+$22], practical privacy concerns extend beyond just transaction details. For instance, public access to account authentication logic makes it easier for malicious actors to design targeted attacks. Consider,

*Work done while the author was an intern at Mysten Labs for example, threshold multi-signature [Ita83], which is a popular authentication mechanism supported by most blockchains for protecting high-value transactions. In threshold multi-signature, n different signing keys are generated such that signatures under at least $t \leq n$ of those keys are needed to authorize a transaction. However, this reveals the access structure (n, t) giving critical information to attackers who learn exactly how many accounts they need to compromise.

In this work, we investigate approaches to conceal the account authentication logic (i.e., the policy) on smart-contract supporting public blockchains such as Ethereum, Solana, Aptos and Sui. A popular solution to hide the access structure is to enforce it *off-chain* with threshold signatures [Des88, DF90] issued by a set of custodians or guardians. In such a threshold authenticator[1], a single key is secret-shared between n parties such that any t parties can generate partial signatures, and aggregate them into a complete signature. Crucially, this final signature looks identical to a signature generated by the original key. Thus, threshold authenticators hide the access structure (both t and n) and inherently obscure the identity of the signers within a group against blockchain transaction observers. This additional privacy makes them a preferable alternative to the aforementioned multi-signature authenticators, as noted by a recent user study [MDM$^+$23].

The need for complex authentication policies. Threshold-wallets (i.e., cryptographic wallets with authentication enforced via threshold signatures), however, do not support the broad range of authentication needs seen in practice. A recent example is that of *account abstraction* [WC23] which allows for programmable access to user accounts via smart contracts instead of relying exclusively on private keys. For one, threshold-wallets—by definition—only support a threshold access structure rather than arbitrary boolean formulae. It is also not possible to combine existing keys created under different signature schemes, so for instance, it is not possible to create a 1-out-of-2 access structure between an ECSDA and an EdDSA key under existing threshold signature schemes. Further, some implementations of threshold EdDSA wallets can inadvertently reveal that the wallet is a threshold-wallet rather than a single key[2]!

In practice, authentication policy designers may want to use transaction data to select an appropriate access structure so as to balance usability and security concerns. Which is to say that they might, for instance, want to use 2-out-of-3 keys for high-value transactions above a certain amount, and only 1-out-of-3 keys otherwise; or they might require authorization from a special administrator key for certain transactions. Rich contextual policies like these are commonly seen in smart-contract based authenticators [Saf24, Arg24] (where the authentication policy is expressed in a smart contract) and off-chain authenticators [Bit24, Fir24] (where the authentication policy is enforced off-chain by a trusted party). Finally, zero-knowledge proofs offer similarly complex authentication policies with some limited degree of privacy for the inputs to the policy, but not the policy itself [Mic24].

To summarize, existing threshold authenticators offer some amount of privacy but only support very specific authentication policies. On the flip side, smart-contract authenticators leverage the rich language support to encode arbitrarily complex policies, but cannot hide the authentication policy; and zero-knowledge proofs offer only limited privacy but for general authentication policies. This is also summarized in Table 1 below.

This observation leads to the central question that motivates this work: *can we build private authenticators that are capable of $\underline{\text{hiding arbitrarily complex policies}}$?*

Our results. To address this question, we design a new class of authentication schemes to validate user transactions on the blockchain while hiding the underlying (authentication) policy. Since they leverage zero-knowledge proof systems as the core building block, we refer to these schemes as *zero-knowledge authenticators* (zkAt). We summarize our contributions below:

❶ **Formalizing zero-knowledge authenticators:** we formalize the notion of zero-knowledge authenticators with policy-privacy for policies expressible as general NP statements. Specifically, the property of policy-privacy guarantees that an adversary learns nothing about the underlying policy besides its public inputs.

[1]An authenticator is simply an authentication mechanism which encodes a general access structure in the form of an authentication policy.

[2]Many threshold EdDSA libraries introduce randomness in signatures on retries. So if different signatures for the same message are observed, it can reveal the use of a threshold-wallet [Kos23].

❷ **Constructing zero-knowledge authenticators:** we give a practical construction for zkAt by building on the seminal work of Groth [Gro16], known colloquially as simply 'Groth16'. More precisely, we define the property of verification-key equivocation and show that non-interactive zero-knowledge proof systems (NIZK) with this property can be generically used to instantiate zkAt. Then, we modify the Groth16 construction so that it satisfies verification-key equivocation, thus yielding a zkAt.

❸ **Oblivious policy updates:** once policy-privacy has been achieved, a second interesting question emerges: *can policies be updated without leaking any new public information?*

As our third contribution, we resolve this question by introducing zkAt$^+$, which extends our standard zkAt by additionally allowing for such oblivious policy updates. We also give a theoretical construction for zkAt$^+$ using recursive NIZKs.

❹ **Demonstrating practicality:** we implement both zkAt and zkAt$^+$, and evaluate them against similar (but less expressive) threshold signatures. Our results demonstrate that zkAt offers comparable performance, even for more complex policy semantics, and as the underlying NIZK is a succinct argument of knowledge (zk-SNARK), the final proof size is independent of the policy.

Type	**Expressiveness**	**Policy anonimity set**	**Oblivious updates**
Multi-signature	Limited	None	N/A
Threshold	Limited	Pre-defined	N/A
Smart-contract	Rich	None	N/A
Zero-knowledge	Rich	Pre-defined	✗
zkAt	Rich	All	✗
zkAt$^+$	Rich	All	✓

Table 1: A comparison between authenticators. "Expressiveness" refers to whether complex policies can be specified. "Policy anonymity set" indicates the level of policy privacy achieved: None (no privacy), a pre-defined policy set (some privacy), all policies (full privacy).

1.1 Application overview

Our stated goal is to build a private authenticator for transactions on public blockchains. More precisely, we want to build an authenticator for arbitrarily complex policies, such that the underlying policy remains hidden from all parties that did not generate the policy and/or the proof. Before delving into the technical details, we explore two scenarios where zkAt can be applied.

❶ **Delegated transactions.** High-level executives at an organization hold shared custody of the organizational finances in the form of a threshold-wallet. The board members of the organization vote on the policies required to authenticate transactions made with the wallet. They might, for instance, require an authentication policy stating that all "large" transactions initiated by the company must be signed by all the executives and at least 50% of the board. For privately-held organizations, it may be in their interest to keep such policies hidden from the public as an added layer of protection for the organizational funds.

❷ **Self-custody solutions.** As another example, consider an individual user who wants to protect their assets using a policy that requires, for instance, the user's valid JSON web tokens (JWT) issued by two-out-of-three OpenID providers (cf. [BCJ$^+$24] for a discussion on JWT and OpenID) the user has previously registered, in order to authenticate a transaction.

In both examples, zkAt makes it more challenging for attackers to mount a successful attack by hiding all information about the authentication logic and access structure. In particular, in the first example, not only do attackers external to the organization not know how many signatures are needed to authenticate a transaction, they do not even know whether this threshold varies by amount and, if so, what the amount is! Similarly, in the second example, the attackers don't know which OpenID providers the user has registered and, by extension, which accounts they need to compromise.

Remark 1.1. We wish to emphasize that, more generally, in the scenarios we envision, the authentication policies are hidden from the validators as well as other (public) third-parties. At first brush, this may seem counter-intuitive—how can a validator validate a transaction without knowing the underlying policy? Our claim is that there is no reason for the validator to know the policy at all! Whatever the policy may be, a validator's only concern is to ensure that the transaction satisfies it, i.e., given the transaction as input, the verification algorithm accepts under the policy issuer's verification key. Importantly, as both our examples illustrate, the policy issuer could be the user (prover) themselves, and it is in their interest to design strong authentication policies.

1.2 Technical overview

To illustrate our technical ideas, we will begin by first considering a simple approach where we only attempt to hide the user's secret credentials.

A simple first approach. The idea is to instantiate a general-purpose zero-knowledge proof system (such as [Gro16, GWC19]) with the policy circuit as input to the setup. As a concrete example, to create a 1-out-of-2 multi-signature between two existing on-chain accounts while also hiding the identities of the two multi-signature participants, a user can proceed in the following manner:

- **Setup.** Create commitments c_1, c_2 to two addresses, i.e., $c_i = \mathsf{Commit}(\mathsf{addr}_i; r_i)$, where each address addr_i is a signature verification key for some signer. Create a designated-prover NIZK (DP-NIZK) proving key and (proof) verification key pair $(\mathsf{pk}, \mathsf{vk})$ for the policy P given by the relation,

$$\left\{ \begin{array}{l} x := (c_1, c_2, \mathsf{tx}_{\mathsf{pb}}), \\ w := (\mathsf{addr}, r, \sigma, \mathsf{tx}_{\mathsf{pv}}) \end{array} : \begin{array}{c} \left(\begin{array}{c} c_1 = \mathsf{Commit}(\mathsf{addr}; r) \vee \\ c_2 = \mathsf{Commit}(\mathsf{addr}; r) \end{array} \right) \\ \wedge\ \mathsf{Sig.Verify}(\mathsf{addr}, \mathsf{tx}_{\mathsf{pb}} || \mathsf{tx}_{\mathsf{pv}}, \sigma) = 1 \end{array} \right\},$$

where $\mathsf{tx}_{\mathsf{pb}}$ (resp. $\mathsf{tx}_{\mathsf{pv}}$) is the public (resp. private) part of the transaction. The proof verification key vk acts as the user's on-chain address.
- **Signing.** Split the transaction into public and private components, as determined by the privacy requirements. Collect signature σ on the full transaction $\mathsf{tx}_{\mathsf{pb}} || \mathsf{tx}_{\mathsf{pv}}$ from one of the two accounts $\mathsf{addr} = \mathsf{addr}_1$ or $\mathsf{addr} = \mathsf{addr}_2$. Generate a ZK proof $\pi = \mathsf{ZK.Prove}(\mathsf{pk}, (c_1, c_2, \mathsf{tx}_{\mathsf{pb}}), (\mathsf{addr}, r, \sigma, \mathsf{tx}_{\mathsf{pv}}))$ proving that the policy is satisfied or, in other words, the signature verifies with one of the two addresses.

Thus the resulting signature is nothing but a zero-knowledge proof, and the privacy guarantee for the private inputs follows straightforwardly from the zero-knowledge property. Notably, the above sketch already permits more expressive policies than threshold signatures albeit without hiding the policy—the fact that P is a 1-out-of-2 authentication policy is known to the public, however the actual addresses remain hidden thanks to the hiding property of the commitment (as long as r_1 and r_2 are secret), as do the private inputs to the circuit.

Next, in order to hide the *policy* itself, the most obvious solution would be to universalize the circuit. For example, we could make the circuit do $n = \mathsf{poly}(\lambda)$ signature verifications irrespective of the policy. This would allow policies that use up to n signatures to use the same circuit. In effect, the "policy anonymity set" consists of all policies that use up to n signatures. While this straw-man approach works to an extent, it results in poor signing performance (even if someone's policy only uses $O(1)$ signatures, they need to verify all n to generate the signature). Moreover, as policies become more expressive, the policy anonymity set grows exponentially (if there are m possible triggers, a universal circuit would have to verify all the 2^m possibilities). Thus the signing overhead for our simple first approach scales proportionally to the size of the policy anonymity set, resulting in an unsatisfactory trade-off between privacy and performance.

Achieving efficient policy privacy. In order to achieve policy-privacy without sacrificing performance, we must ask a more fundamental question about our sketch above (without the universalized circuit): assuming that the proving key is held privately by the user, do the verification key and proof reveal any non-trivial information about the policy?

A priori, it is not at all obvious whether a zkAt is directly constructible from any existing NIZK schemes. So, as a first step, we must formalize our, presently abstract, policy-privacy property. To that end, we define a new property for NIZK proof systems, which we call *verification key equivocation* (vk-equivocation). At a high level, in the vk-equivocation experiment, an adversary—holding a proof verification key—must identify which of the two policies (of its choice) was an honest proof created with respect to, for a statement (also of its choice) that satisfies both policies. The crucial observation is that if the underlying NIZK has equivocable verification keys, or in other words if the verification key is independent of the underlying relation (which encodes the policy), then a zkAt (instantiated according to our simple first approach) hides the policy since the proof is already zero-knowledge!

Our next task then is to develop such a proof system with vk-equivocation. For this, we turn to the work of Groth [Gro16], that describes a NIZK for quadratic arithmetic programs (QAPs) from pairing-based assumptions (cf. § 2.4 for a description of the scheme). In particular, we find that a simple modification to Groth16 is sufficient to achieve this property. At a high level, our main observation is that the only component linking the verification key to the underlying relation are the evaluations of the QAP polynomials at a random point x; and as it turns out, we can interpolate fresh polynomials that behave exactly as the original ones on the characteristic points (so that they describe the same arithmetic circuit), but additionally, also evaluate at x to *a priori* uniformly chosen values. These fresh polynomials now define an updated QAP. This essentially fixes the evaluation of the polynomials at x independently of the relation, and consequently the resulting verification key is made completely independent of the updated QAP. Most notably, this modification affects no change to the proof verification function, thus making it fully compatible with existing Groth16 verifiers!

The overall workflow for setting up zkAt keys should thus be:

1. Choose a policy and encode it into a circuit,
2. Run the modified Groth16 setup to generate the trapdoor, proving and verification keys,
3. Delete the trapdoor (as we explain shortly, if storing a sensitive secret is acceptable and oblivious policy updates are desired, then the trapdoor can be persisted); and
4. Store the proving key, and publish the verification key as the on-chain address.

Interestingly, unlike in most other use cases of Groth16, the presence of a trusted setup phase is not a problem. This is because it is in the interest of the user, who is the trusted setup generator, to delete the trapdoor safely as otherwise their account could be compromised.

Updating policies obliviously. Policy issuers may want to be able to update their policies for any number of reasons, such as for operational reasons such as periodic key rotation. A trivial way to do this, of course, is to issue a fresh set of keys with respect to the new policy. However, recall that the verification key acts as the user's on-chain address, and thus the trivial approach would require updating user addresses every time a policy is changed. It is therefore desirable to grant policy issuers the ability to update policies without having to change the verification keys. We call this feature *oblivious updateability.*

Interestingly, our Groth16-zkAt already achieves oblivious updateability in a limited sense—the idea is to retain the trapdoor for our modified proof system (in cold storage), so that new proving keys can be generated at will. Old policies and the corresponding proving keys can then be retired by adding a clause within the policy circuit to check that the current time is less than a fixed expiry time (this assumes that the current time is accessible as a public input, a feature commonly available on most blockchains).

However, persisting the trapdoor carries significantly more risk as a leaked trapdoor breaks security. Moreover, we only know our Groth16-zkAt to be securely updateable assuming that an attacker cannot access two different proving keys corresponding to the same verification key, but in situations such as oblivious policy updates, this is not necessarily the case—a user (prover) with two proving keys for the same verification key can learn non-trivial infomation about the trapdoor and possibly break soundness of the NIZK.

Maliciously-secure oblivious policy updates. The upshot of this is that we must extend our zkAt definition so as to realize maliciously-secure policy updates in the strongest sense, i.e., one where the adversary is given access to an update oracle, which returns updated proving keys (corresponding to the same verification key) for policy updates of the adversary's choice. We call the extended primitive that satisfies the stronger update security, zkAt^{+}.

The main technical challenge in constructing a zkAt$^+$ is to somehow fix the proof verification key across policy updates since it acts as the user's on-chain address while *securely* updating the proving key. As we just explained, however, the Groth16-zkAt approach does not quite work, so we approach this problem from a new direction. As usual, during authentication, the user will still compute a proof π_I that the transaction satisfies the underlying policy using a (not necessarily designated-prover) NIZK. The astute reader may observe that the user cannot send this proof in the clear, since it obviously contains information about the policy. Indeed, instead the user recursively composes this proof with another "outer" NIZK proof π_O, essentially proving that it has a proof that the transaction satisfies the policy. Notably, as this outer NIZK has the same structure for any policy, there need only be a single global common reference string (CRS), crs_O that can be used to generate and verify outer proofs for *all* users.

Observe, also, that the policy can now be updated obliviously, since crs_O does not depend on any policy specific information. Unfortunately, this construction is incomplete, since there are no clear candidate public keys that could play the role of a user address. One might again be tempted to set the "inner" NIZK proof's verification key as the user's address, but remember that this is not possible if we want oblivious updates. Instead of a proof verification key acting as the user's on-chain address, we let it be given by a *signature* verification key vk, for a signing-verification key pair $(\mathsf{sk}, \mathsf{vk})$ generated by the user. Now, in addition to the inner proof π_I, the user must additionally compute a signature σ on the inner proof's CRS, crs_I, and then, in the outer proof, also prove that σ verifies under vk. Thus, the public instance for the outer NIZK is the address vk along with any other public transaction data, and the private witness is crs_I, π_I and σ.

In order to update a policy, the policy issuer simply computes a fresh crs_I for the new policy and signs it with its secret signing key sk. Clearly, this reveals no information regarding the update on the chain. Moreover, this construction has maliciously-secure policy updates by soundness of the both NIZKs.

We remark that in applications where the policy issuers are distinct from the users (which, recall, need not always be the case) there does arise a subtle issue with this approach, namely that a user holding an older proving key can still authenticate with respect to that policy. This, however, is easily circumvented by having the policy issuer additionally sign an arbitrarily chosen tag that the user must prove belongs to a public set of currently accepted tags, and has the benefit of allowing policies to expire gracefully.

1.3 Related Work

Existing blockchain authentication methods. Threshold [Des88, DF90, KG20, BCK$^+$22, RRJ$^+$22, ANO$^+$22] and multi-signing [MOR01, BDN18] are commonly used authentication mechanisms in blockchain settings. Our solution offers all the same benefits of threshold-based solutions such as privacy of signers, compact signatures while also capturing more complex policy semantics beyond the threshold. Moreover, with our zkAt, one can do this with *pre-existing* signing keys and without requiring any expensive distributed key-generation, a pre-requisite for threshold signatures. Smart-contract based authenticators [Arg24, Saf24] (sometimes called account abstraction wallets [WC23]) are popular for their flexibility and security features, for example, account recovery, flexible policies for high-value transactions, ability to change policies over time. However, these offer no notion of privacy. Interestingly, zkAt can be used to turn any smart-contract based authenticator private.

Attribute-based authentication. Another common authentication mechanism is based on user attributes satisfying certain pre-specified criteria [MPR11, LAS$^+$10]. Indeed, this is nothing but a policy-based authentication mechanism with the important distinction that the authentication policy need not be private. Put another way, one may view zkAt as an extension of attribute-based authentication that offers stronger privacy guarantees for not just the user's attributes, but also the constraints on those attributes.

Functional commitments. A functional commitment scheme [LRY16] enables a user to succinctly commit to a function (from a specified family), such that the user can later verifiably reveal values of the function at desired inputs. Such a commitment must be *binding* to the function and may additionally also *hide* [BNO21] it. We observe that zkAt realizes a sort of function-private functional commitment scheme for functions with binary outputs, with vk being the commitment to the policy function. Perhaps for the first time, our equivocable Groth16 construction gives a functional commitment where the underlying function is updateable or equivocable (given the trapdoor) without changing the commitment.

2 Preliminaries

In this section, we provide preliminaries needed in this work with more deferred to Appendix A.

Notation. Let λ denote the security parameter, and PPT denote probabilistic polynomial-time. We use $\leftarrow\$$ to denote the output of a randomized algorithm, $\leftarrow$ to denote output of a deterministic algorithm, and $:=$ for assignment. For our security definitions, we use notation similar to [Gro16].

Following common convention we use lowercase bold-face letters to denote vectors and uppercase bold-face letters for matrices. For a vector $\mathbf{x}$, x_i denotes its i^{th} element. In general $\mathbb{G}$ denotes a group and $\mathbb{F}$ a field. Let g_i be the generator of a group $\mathbb{G}_i$, then we write g_i^x for $x \in \mathbb{F}$ as $[x]_i$ and $a[x]_i := g_i^{ax}$, for some $a \in \mathbb{F}$. As usual, $\mathbb{Z}$ is the ring of integers and $\mathbb{Z}_p$ is the ring of integers modulo p for some integer $p > 0$. Finally, $\mathbb{Z}_p[X]$ is the ring of polynomials with coefficients in $\mathbb{Z}_p$, and for any polynomial $U(X) \in \mathbb{Z}_p[X]$ the notation $\deg(U)$ denotes its degree.

2.1 Bilinear Pairings

Definition 2.1 (Bilinear pairings). *Given a security parameter λ, a bilinear group generator $\mathsf{BG}(1^\lambda)$ returns a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, q, g_1, g_2)$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of the same prime order q with the generators g_1, g_2 respectively. Also, let $\mathbb{Z}_q$ be the field of order q. A bilinear pairing is an efficiently computable map, $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, satisfying the following properties:*

- **Bilinearity:** $\forall P \in \mathbb{G}_1, Q \in \mathbb{G}_2,\ a, b \in \mathbb{Z}_q$,

$$\mathsf{e}(P^a, Q)^b = \mathsf{e}(P, Q^b)^a = \mathsf{e}(P, Q)^{ab} \quad .$$

- **Non-degeneracy:** $\mathsf{e}(g_1, g_2) \neq 1_{\mathbb{G}_T}$.

Using the notation from [Gro16], we denote $\mathsf{e}(g_1^a, g_2^b) =: [a]_1 \cdot [b]_2 = [ab]_T$.
Bilinear pairings can be of a few types depending on whether there is an efficient isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ in both directions (Type 1), only one direction (Type 2), or in neither direction (Type 3) [GPS06]. Type 3 pairings are the most efficient setting for a relevant security parameter.

2.2 Cryptographic Building Blocks

We recall the definitions for some common cryptographic primitives that we use in our constructions.

2.2.1 Signature schemes

A signature scheme Sig over message space $\mathcal{M}$ consists of the following polynomial time algorithms:

- $\underline{\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})}$. is a randomized algorithm that takes security parameter λ as input and returns a pair of keys $(\mathsf{vk}, \mathsf{sk})$, where vk is the verification key and sk is the signing key.
- $\underline{\mathsf{Sign}(\mathsf{sk}, M) \to \sigma}$. is a possibly randomized algorithm that takes as input the signing key sk, and a message $M \in \mathcal{M}$, and returns a signature σ.
- $\underline{\mathsf{Verify}(\mathsf{vk}, M, \sigma) \to \{0, 1\}}$. is a deterministic algorithm that takes as input the verification key vk, a message $M \in \mathcal{M}$, and a signature σ. It outputs 1 (accept) or 0 (reject).

A signature scheme satisfies correctness if for all $\lambda \in \mathbb{N}$, $M \in \mathcal{M}$, and every signing-verification key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, every signature $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, M)$, $\mathsf{Verify}(\mathsf{vk}, M, \sigma) = 1$.

Definition 2.2 (Existential unforgeability under chosen messages)**.** *A signature scheme* $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *is existentially unforgeable under chosen messages if for every* PPT *attacker* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *the following probability is at most* $\epsilon(\lambda)$

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, M^*, \sigma^*) = 1 \;:\; \begin{array}{l}(\mathsf{vk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda) \\ (M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}}(\mathsf{vk})\end{array}\right]$$

and $\mathcal{A}$ *should never have queried* M^* *to the signing oracle,* $\mathcal{O}_{\mathsf{sk}}(\cdot)$*.*

2.2.2 Non-interactive zero-knowledge

Let $\mathsf{C}_\lambda \coloneqq \left\{\mathsf{C} : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}\right\}$ be a family of boolean circuits computable in polynomial time. Then, a NIZK proof system for a circuit C_λ consists of the following polynomial time algorithms:

- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{C}) \to (\mathsf{crs}, \tau)}$. The setup algorithm takes as input the security parameter λ and a circuit C, and outputs a common reference string crs.
- $\underline{\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi}$. The prover algorithm takes as input a crs, an instance x, and a witness w. It outputs a proof π.
- $\underline{\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1}$. The verification algorithm takes as input a crs, an instance x, and a proof π. It outputs 1 (accept) or 0 (reject).

Definition 2.3 (Non-interactive Zero-knowledge)**.** *Given a circuit* C*, a NIZK proof system for an* NP *relation* $\mathcal{R} = \{(x, w) : \mathsf{C}(x||w) = 1\}$ *must satisfy the following properties:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, and every crs computed as* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C})$*, any instance and witness pair* $(x, w) \in \mathcal{R}$*,*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \;:\; \pi \leftarrow\!\!\$\ \mathsf{Prove}(\mathsf{crs}, x, w)\right] = 1.$$

- **Soundness:** *For every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *the following probability is at most* $\epsilon(\lambda)$*,*

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\ x \notin \mathcal{L}_{\mathcal{R}}\end{array} \;:\; \begin{array}{l}\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})\end{array}\right]$$

- **Zero-knowledge:** *There exists a* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *for every adversary* $\mathcal{A}$ *and such that there is a negligible function* $\epsilon(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every* $(x, w) \in \mathcal{R}$ *the following probability is at most* $\epsilon(\lambda)$*,*

$$\left|\Pr\left[\begin{array}{l}\mathcal{A}(\mathsf{crs}, x, \pi) = 1 : \\ \qquad\qquad \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \qquad\qquad \pi \leftarrow\!\!\$\ \mathsf{Prove}(\mathsf{crs}, x, w)\end{array}\right] - \Pr\left[\begin{array}{l}\mathcal{A}(\mathsf{crs}, x, \pi) = 1 : \\ \qquad\qquad (\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{C}) \\ \qquad\qquad \pi \leftarrow \mathcal{S}_2(\tau, x)\end{array}\right]\right|$$

Argument of Knowledge. Further, a NIZK proof system is an *argument of knowledge* if it satisfies the following additional property:

- **(Computational) Knowledge soundness:** For every PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ and a negligible function $\epsilon(\cdot)$, such that for all $\lambda \in \mathbb{N}$ the following probability is at most $\epsilon(\lambda)$,

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{crs}, x^*, \pi^*) = 1 \\ \wedge\ (x^*, w^*) \notin \mathcal{R}\end{array} \;:\; \begin{array}{l}\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ ((x^*, \pi^*); w^*) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{crs})\end{array}\right]$$

where the notation $((x^*, \pi^*); w^*) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{crs})$ is shorthand for $(x^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{crs})$ and $w^* \leftarrow \mathcal{E}(x^*, \pi^*)$.

Designated-prover NIZK. A (publicly verifiable) designated-prover NIZK scheme (DP-NIZK) for a circuit C consists of the following polynomial time algorithms:

• $\underline{\mathsf{Setup}(1^\lambda, \mathsf{C}) \to (\mathsf{vk}, \mathsf{pk})}$. The setup algorithm takes as input the security parameter λ and a boolean circuit C, and outputs a verification key vk and a proving key pk.
• $\underline{\mathsf{Prove}(\mathsf{pk}, x, w) \to \pi}$. The prover algorithm takes as input the proving key pk, an instance x, and a witness w. It outputs a proof π.
• $\underline{\mathsf{Verify}(\mathsf{vk}, x, \pi) \to 0/1}$. The verification algorithm takes as input a verification key vk, an instance x, and a proof π. It outputs 1 (accept) or 0 (reject).

A DP-NIZK scheme must further satisfy the same properties as described in Definition 2.3 with the crs appropriately replaced by pk and vk.

2.3 Quadratic arithmetic programs

A quadratic arithmetic program (QAP) comprises of a finite field $\mathbb{Z}_p$ for some prime p with $|p| = \lambda$, integers $\ell \le m$, and polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^m$ and $T(X)$ in $\mathbb{Z}_p[X]$ with $\deg(U_i), \deg(V_i), \deg(W_i) < \deg(T) = n$ (for all $i \in [0, m]$) such that, for $a_0 := 1$, it defines the following relation

$$\left\{ \begin{array}{l} x := (a_i)_{i\in[0,\ell]} \in \mathbb{Z}_p^\ell \\ w := (a_i)_{i\in[\ell+1,m]} \in \mathbb{Z}_p^{m-\ell} \quad : \\ \qquad \sum_{i=0}^m a_i U_i(X) \cdot \sum_{i=0}^m a_i V_i(X) = \sum_{i=0}^m a_i W_i(X) \mod T(X) \end{array} \right\}$$

In this work, we consider proof systems for satisfiability of general arithmetic circuits, which consist of addition and multiplication gates over the finite field $\mathbb{Z}_p$. Gennaro, et al. [GGPR13] gave an efficient technique for converting any arithmetic circuit into a QAP, thus allowing us to prove statements encoded as general arithmetic circuits using Groth16.

2.4 Recalling Groth16

Since it will be essential to one of our constructions, we now recall the Groth's NIZK argument for QAPs (and therefore for any arithmetic circuit).

Now, for some prime p such that $|p| = \lambda$, groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ such that the pairing $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. Consider a QAP,

$$\mathscr{R} = \left\{ p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2, \ell, \{U_i(X), V_i(X), W_i(X)\}_{i=0}^m, T(X) \right\} ,$$

that defines a field $\mathbb{Z}_p$ and a language of statements $(a_1, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ and witnesses $(a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}$ such that for $a_0 = 1$, polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^m$ and $T(X)$ in $\mathbb{Z}_p[X]$ with $\deg(U_i)$, $\deg(V_i), \deg(W_i) < \deg(T) = n$ (for all $i \in [0, m]$) and $T(X) := \prod_{i=1}^n (X - r_i)$ for all distinct $r_i \in \mathbb{Z}_p^*$, the following holds

$$\sum_{i=0}^m a_i U_i(X) \cdot \sum_{i=0}^m a_i V_i(X) - \sum_{i=0}^m a_i W_i(X) = H(X)T(X)$$

for some degree $(n-2)$ polynomial $H(X) \in \mathbb{Z}_p[X]$. Then, the Groth16 NIZK argument is given by the following algorithms:

• **Setup.** The setup algorithm accepts the QAP $\mathscr{R}$ as input, sets the secret trapdoor $\tau := (\alpha, \beta, \gamma, \delta, x)$ for $\alpha, \beta, \gamma, \delta, x \leftarrow\$ \mathbb{Z}_p^*$, and outputs it along with the verifier key vk and the prover key pk where

$$\mathsf{vk} := \begin{pmatrix} [\alpha]_1,\ [\beta]_2,\ [\gamma]_2,\ [\delta]_2, \\ \left\{[\chi_i]_1 := \left[\frac{\beta U_i(x)+\alpha V_i(x)+W_i(x)}{\gamma}\right]_1\right\}_{i=0}^{\ell} \end{pmatrix},$$

$$\mathsf{pk} := \begin{pmatrix} [\alpha]_1,\ [\beta]_1,\ [\beta]_2,\ [\delta]_1,\ [\delta]_2,\ \left\{[\theta_j]_1 := \left[\frac{x^j T(x)}{\delta}\right]_1\right\}_{j=0}^{n-2} \\ \{[\psi_i]_1 := [U_i(x)]_1\}_{i=0}^{m},\ \{[\varphi_i]_2 := [V_i(x)]_2\}_{i=0}^{m}, \\ \left\{[\zeta_i]_1 := \left[\frac{\beta U_i(x)+\alpha V_i(x)+W_i(x)}{\delta}\right]_1\right\}_{i=\ell+1}^{m} \end{pmatrix}.$$

• **Prove.** The proving algorithm samples $r, s \leftarrow \mathbb{Z}_p^*$ and, using the instance and witness $(a_1, \dots, a_\ell, a_{\ell+1}, \dots, a_m) \in \mathbb{Z}_p^m$, sets the polynomial

$$H(X) := \frac{\sum_{i=0}^{m} a_i U_i(X) \cdot \sum_{i=0}^{m} a_i V_i(X) - \sum_{i=0}^{m} a_i W_i(X)}{T(X)}.$$

It then computes

$$\pi := \begin{pmatrix} [A]_1 := [\alpha]_1 + r[\delta]_1 + \sum_{i=0}^{m} a_i[\psi_i]_1, \\ [B]_2 := [\beta]_2 + s[\delta]_2 + \sum_{i=0}^{m} a_i[\varphi_i]_2, \\ [C]_1 := \begin{pmatrix} s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1 + \\ \sum_{i=\ell+1}^{m} a_i[\zeta_i]_1 + \sum_{j=0}^{n-2} h_j[\theta_j]_1 \end{pmatrix} \end{pmatrix} \tag{1}$$

given the pk, and outputs π as the proof.

• **Verify.** Given the vk, the instance $(a_1, \dots, a_\ell) \in \mathbb{Z}_p^\ell$ and a proof $\pi := ([A]_1, [B]_2, [C]_1)$, the verifier simply checks whether:

$$[A]_1 \cdot [B]_2 \stackrel{?}{=} [\alpha]_1 \cdot [\beta]_2 + [C]_1 \cdot [\delta]_2 + \left(\sum_{i=0}^{\ell} a_i[\chi_i]_1\right) \cdot [\gamma]_2$$

and outputs the result.

3 Zero-knowledge Authenticators with Policy-privacy

We now formalize the notion of a zero-knowledge authenticator for a family of authentication policies and construct such an authenticator under a mild assumption that the underlying NIZK has equivocable (verification) keys, a property which we also define.

3.1 Zero-knowledge authenticator

We define a zero-knowledge authenticator for a family of authentication policies. As mentioned in the introduction, a zkAt can easily capture low-level semantics of an authentication mechanism as a (polynomial-size) circuit.

Let $\Pi = \left\{f_\lambda : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}\right\}$ be a family of authentication policies, then a *zero-knowledge authenticator* for any policy $\mathsf{P} \in \Pi$ over the message space $\mathcal{M}$ and private input space Ω consists of the following polynomial time algorithms:

• $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})}$. The setup algorithm takes as input the security parameter λ and the authentication policy P. It outputs a public verification key vk_P and a secret proving key[3] pk_P.
• $\underline{\mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega) \to \pi/\bot}$. The proving algorithm takes as input the secret key pk_P, a message $M \in \mathcal{M}$ to be signed and some private input $\omega \in \Omega$. It outputs a proof π or $\bot$.
• $\underline{\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) \to \{0,1\}}$. The verification algorithm takes as input the public verification key vk_P, a message $M \in \mathcal{M}$, and a proof π. It outputs 0 or 1.

[3]We can assume that pk_P also implicitly contains P.

Definition 3.1 (Zero-knowledge Authenticator). *A zero-knowledge authenticator must satisfy the following properties with respect to any policy* $\mathsf{P} \in \Pi$*:*

- **Completeness:** *For every message* $M \in \mathcal{M}$ *and for every string* $\omega \in \Omega$ *such that* $\mathsf{P}(M||\omega) = 1$*,*

$$\Pr\left[\mathsf{AuthVfy}(\mathsf{vk_P}, M, \pi) = 1 \ : \ \begin{array}{r} (\mathsf{vk_P}, \mathsf{pk_P}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ \pi \leftarrow\!\$\ \mathsf{AuthProve}(\mathsf{pk_P}, M, \omega) \end{array} \right] = 1$$

- **Knowledge soundness***: For every* PPT *adversary* $\mathcal{A}$ *there exists a* PPT *extractor* $\mathcal{E}$ *and a negligible function* $\epsilon(\cdot)$ *satisfying, for all* $\lambda \in \mathbb{N}$ *the following probability is at most* $\epsilon(\lambda)$

$$\Pr\left[\begin{array}{l} \mathsf{AuthVfy}(\mathsf{vk_P}, M^*, \pi^*) = 1 \\ \wedge\ \mathsf{P}(M^*||\omega^*) \neq 1 \end{array} : \begin{array}{r} (\mathsf{vk_P}, \mathsf{pk_P}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ ((M^*, \pi^*);\ \omega^*) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{vk_P}, \mathsf{pk_P}) \end{array} \right]$$

- **(Perfect) zero-knowledge***: There exists a* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for every* PPT *adversary* $\mathcal{A}$*, every* $\lambda \in \mathbb{N}$*, every* $M \in \mathcal{M}$ *and every string* $\omega \in \Omega$ *such that* $\mathsf{P}(M||\omega) = 1$*,*

$$\Pr\left[\begin{array}{r} \mathcal{A}(\mathsf{vk_P}, M, \pi) = 1\ : \\ (\mathsf{vk_P}, \mathsf{pk_P}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ \pi \leftarrow\!\$\ \mathsf{AuthProve}(\mathsf{pk_P}, M, \omega) \end{array} \right] = \Pr\left[\begin{array}{r} \mathcal{A}(\mathsf{vk_P}, M, \pi) = 1\ : \\ (\mathsf{vk_P}, \mathsf{pk_P}, \tau) \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{P}) \\ \pi \leftarrow \mathcal{S}_2(\tau, M) \end{array} \right]$$

- **Policy privacy***: For every stateful* PPT *adversary* $\mathcal{A}$*, there is a negligible function* $\epsilon(\cdot)$ *such that the following probability is at most* $1/2 + \epsilon(\lambda)$

$$\Pr\left[\begin{array}{l} \begin{pmatrix} \forall \hat{b} \in \{0,1\}\ : \\ \mathsf{P}_{\hat{b}}(M^*||\omega^*_{\hat{b}}) = 1 \end{pmatrix} \\ \wedge\ \mathcal{A}(\pi) = b \end{array} : \begin{array}{r} \{0,1\} \leftarrow\!\$\ b \\ \mathsf{P}_0, \mathsf{P}_1 \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{vk}_{\mathsf{P}_b}, \mathsf{pk}_{\mathsf{P}_b}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}_b) \\ (M^*, \omega^*_0, \omega^*_1) \leftarrow \mathcal{A}(\mathsf{vk}_{\mathsf{P}_b}) \\ \pi \leftarrow\!\$\ \mathsf{AuthProve}(\mathsf{pk}_{\mathsf{P}_b}, M^*, \omega^*_b) \end{array} \right]$$

Before giving a formal construction, we must define a new object called designated-prover NIZK schemes (DP-NIZK) with *equivocable verification keys.* At a high level, the property of verification-key equivocation guarantees that the verification key of a DP-NIZK scheme is independent of its circuit. Looking ahead to our construction, when instantiated with a DP-NIZK with this property, we will be able to reduce the policy privacy of our zkAt to the verification key equivocation of the DP-NIZK.

3.1.1 Verification-key equivocation

Informally, the vk-equivocation game models and adversary who, given a verification key and a proof for a statement in languages specified by both circuits of its choice, must decide which of the said circuits does the key (and proof) correspond to.

Definition 3.2 (vk-equivocation). *A publicly verifiable DP-NIZK scheme has equivocable verification keys if for every stateful* PPT *adversary* $\mathcal{A}$*, there is a negligible function* $\epsilon(\cdot)$ *such that the following probability is at most* $\frac{1}{2} + \epsilon(\lambda)$*,*

$$\Pr\left[\begin{array}{l} \forall \hat{b} \in \{0,1\}\ :\ \mathsf{C}_{\hat{b}}(x^*||w^*_{\hat{b}}) = 1 \\ \wedge\ \mathcal{A}(\pi) = b \end{array} : \begin{array}{r} \{0,1\} \leftarrow\!\$\ b \\ \mathsf{C}_0, \mathsf{C}_1 \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{vk}_b, \mathsf{pk}_b) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}_b) \\ (x^*, w^*_0, w^*_1) \leftarrow \mathcal{A}(\mathsf{vk}_b) \\ \pi \leftarrow\!\$\ \mathsf{Prove}(\mathsf{pk}_b, x^*, w^*_b) \end{array} \right]$$

where each circuit C_b *encodes an* NP *relation* $\mathcal{R}_b = \big\{(x, w)\ :\ \mathsf{C}_b(x||w) = 1\big\}$

3.2 Construction

We now provide our construction. It requires a publicly-verifiable DP-NIZK scheme *with* vk-*equivocation* $\mathsf{NIZK} = (\mathsf{NIZK.Setup},$
$\mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for the relation $\mathscr{R}_{\mathsf{P}} = \{(x, w) : \mathsf{P}(x||w) = 1\}$.

- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})}$. Give the security parameter λ and the policy P as input, the setup algorithm runs the NIZK setup to obtain the verification and the prover keys, i.e., $(\mathsf{vk}_{\mathsf{zk}}, \mathsf{pk}_{\mathsf{zk}}) \leftarrow\$ \mathsf{NIZK.Setup}(1^\lambda, \mathsf{P})$. It then outputs $\mathsf{vk}_\mathsf{P} \coloneqq \mathsf{vk}_{\mathsf{zk}}$, and $\mathsf{pk}_\mathsf{P} \coloneqq \mathsf{pk}_{\mathsf{zk}}$.
- $\underline{\mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega) \to \pi/\bot}$. It parses pk_P to obtain $\mathsf{pk}_{\mathsf{zk}}$ and then computes the proof $\pi \leftarrow\$ \mathsf{NIZK.Prove}(\mathsf{pk}_{\mathsf{zk}}, x \coloneqq M, w \coloneqq \omega)$ and outputs it.
- $\underline{\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) \to \{0, 1\}}$. It returns the output of $\mathsf{NIZK.Verify}(\mathsf{vk}_\mathsf{P}, M, \pi)$.

Security. We claim that construction 3 is a zkAt. The knowledge soundness and zero-knowledge properties of zkAt follow directly from the underlying DP-NIZK so we only focus on policy privacy, which we claim follows when the DP-NIZK proof has equivocable verification keys. The formal theorem statement and proof are presented in Appendix B.

4 An Equivocable Groth16

In Section 3.1.1 we defined proofs with equivocable verification keys that are required to instantiate our zkAt construction. We now explain how to concretely build this primitive from the DP-NIZK of Groth [Gro16].

A crucial observation towards achieving policy-privacy is that the Groth16 verification key can be equivocated—in the sense that one can perform the Groth16 setup in a way that guarantees that vk can be sampled independently of the relation. We give a bootstrapping compiler to build an equivocable Groth16 scheme given the non-equivocable version.

Now, for some prime p such that $|p| = \lambda$, groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ such that the pairing $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, consider a QAP,

$$\mathscr{R} = \Big\{p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2, \ell, \{U_i(X), V_i(X), W_i(X)\}_{i=0}^m, T(X)\Big\} \quad , \tag{2}$$

that defines a field $\mathbb{Z}_p$ and a language of statements $(a_1, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ and witnesses $(a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}$ such that for $a_0 = 1$, polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^m$ and $T(X)$ in $\mathbb{Z}_p[X]$ with $\deg(U_i)$, $\deg(V_i), \deg(W_i) < \deg(T) = n$ (for all $i \in [0, m]$) and $T(X) \coloneqq \prod_{i=1}^n (X - r_i)$ for all distinct $r_i \in \mathbb{Z}_p^*$, the following holds

$$\sum_{i=0}^m a_i U_i(X) \cdot \sum_{i=0}^m a_i V_i(X) - \sum_{i=0}^m a_i W_i(X) = H(X)T(X)$$

for some degree $(n-2)$ polynomial $H(X) \in \mathbb{Z}_p[X]$. Then, we have the following constructive procedure:

1. Sample $x, \{y_{U,i}\}_{i=0}^m, \{y_{V,i}\}_{i=0}^m, \{y_{W,i}\}_{i=0}^m \leftarrow\$ \mathbb{Z}_p^*$.
2. For every symbol $S \in \{U, V, W\}$ and for every $i \in [0, m]$ interpolate the polynomial $\tilde{S}_i(X) \in \mathbb{Z}_p[X]$ over coordinates

$$\tilde{S}_i(x) = y_{S,i} \;\text{ and }\; \forall j \in [n] \;:\; \tilde{S}_i(r_j) = S_i(r_j) = S_{i,j} \quad .$$

Given this, we claim that the modified QAP:

$$\tilde{\mathscr{R}} = \Big\{p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2, \ell, \Big\{\tilde{U}_i(X), \tilde{V}_i(X), \tilde{W}_i(X)\Big\}_{i=0}^m, T(X)\Big\} \tag{3}$$

defines the same relation as $\mathscr{R}$. Formally,

Lemma 4.1. *$T(X)$ divides $\sum_{i=0}^m a_i\tilde{U}_i(X) \cdot \sum_{i=0}^m a_i\tilde{V}_i(X) - \sum_{i=0}^m a_i\tilde{W}_i(X)$ if and only if $(a_1, \ldots, a_m)$ is a satisfying assignment for $\mathscr{R}$.*

Please refer to Appendix C for the proof of this lemma.

In short, by interpolating fresh polynomials $\left\{\tilde{U}_i, \tilde{V}_i, \tilde{W}_i\right\}_i$ that behave exactly as $\{U_i, V_i, W_i\}_i$ (respectively, for each i), but also evaluate at x to uniformly chosen values $\{y_{U,i}, y_{V,i}, y_{W,i}\}_i$ (respectively, for each i), we have essentially fixed the evaluation of the polynomials at x independently of the relation. The result is that the verification key generated during setup will now be made completely independent of the updated QAP which, as we have just shown, defines the same relation as the original QAP. Accordingly, let us now define the modified setup algorithm (prove and verify algorithms do not change):

• **Setup.** The setup algorithm accepts a QAP $\mathscr{R}$ as explained by (2), sets the secret trapdoor:

$$\tau := (\alpha, \beta, \gamma, \delta, x, \{y_{U,i}\}_{i=0}^m, \{y_{V,i}\}_{i=0}^m, \{y_{W,i}\}_{i=0}^m)$$

for all values in τ sampled uniformly from $\mathbb{Z}_p^*$, and outputs it along with the verifier key vk and the prover key $\left(\mathsf{pk}, \tilde{\mathscr{R}}\right)$ where $\tilde{\mathscr{R}}$ is of the form described in (3) and,

$$\mathsf{vk} := \left(\begin{array}{c} [\alpha]_1,\ [\beta]_2,\ [\gamma]_2,\ [\delta]_2, \\ \left\{[\chi_i]_1 := \left[\frac{\beta y_{U,i} + \alpha y_{V,i} + y_{W,i}}{\gamma}\right]_1\right\}_{i=0}^{\ell} \end{array}\right),$$

$$\mathsf{pk} := \left(\begin{array}{c} [\alpha]_1,\ [\beta]_1,\ [\beta]_2,\ [\delta]_1,\ [\delta]_2,\ \left\{[\theta_j]_1 := \left[\frac{x^j T(x)}{\delta}\right]_1\right\}_{j=0}^{n-2} \\ \left\{[\psi_i]_1 := [y_{U,i}]_1\right\}_{i=0}^m,\ \left\{[\varphi_i]_2 := [y_{V,i}]_2\right\}_{i=0}^m, \\ \left\{[\zeta_i]_1 := \left[\frac{\beta y_{U,i} + \alpha y_{V,i} + y_{W,i}}{\delta}\right]_1\right\}_{i=\ell+1}^m \end{array}\right).$$

Security. The main theorem for this construction states that it is a NIZK proof system with vk-equivocation. In addition, completeness and zero-knowledge follow straightforwardly, so we only show that this construction is also knowledge-sound. This is described and proven formally in Appendix C.

Remark 4.2. Observe that our zkAt from § 3, instantiated with the modified Groth16 is *updatable* in the following sense: given the trapdoor τ, an honest user can update the policy to a different relation $\mathscr{R}'$, while keeping the same vk. In fact, the only change needed is in the QAP. This updatability makes sense, for example, in single user scenarios, where the user wants to be able to update their own policies without changing the address (vk). The trapdoor can be kept in cold storage and only recalled when updating the policy. However, the assumption of an honest prover is insufficient for delegation such as when, for example, honest Alice wants to delegate transaction signing capabilities to (potentially malicious) Bob under some policy, and then later decides to update the policy. The Groth16-zkAt fails to achieve full update security in this case because Bob can gain non-trivial knowledge about the trapdoor τ if given two proving keys. We address the stronger security requirement of secure updates against malicious provers in the next section.

5 Obliviously Updateable Policy-privacy

In this section, we introduce zkAt with *oblivious updates* wherein, given a $\mathsf{pk_P}$ with respect to an existing policy $\mathsf{P} \in \Pi$, a policy issuer can update $\mathsf{pk_P}$ to $\mathsf{pk'_P}$ for some new policy $\mathsf{P}' \in \Pi$ without updating the corresponding $\mathsf{vk_P}$. Following the formal definition, we give a generic construction for zkAt using recursive NIZKs.
An *obliviously updateable* zkAt, that we call zkAt$^+$, additionally consists of the following polynomial time algorithms:

• $\underline{\mathsf{Gen}(1^\lambda) \to \mathsf{pp}}$. The parameter generation algorithm as input the security parameter λ, and outputs public parameters pp for the protocol. All algorithms of zkAt$^+$ take pp as input, but we omit it for brevity. This algorithm is run once and for all.

• $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \rightarrow (\mathsf{vk_P}, \mathsf{pk_P}, \kappa)}$. The setup algorithm takes as input the security parameter λ and the authentication policy P. It outputs a public verification key $\mathsf{vk_P}$, a secret proving key $\mathsf{pk_P}$ and a secret update key κ.
• $\underline{\mathsf{PolUpdate}(\mathsf{pk_P}, \kappa, \mathsf{P}') \rightarrow \mathsf{pk}'_\mathsf{P}/\bot}$. The policy update algorithm takes as input the secret proving key $\mathsf{pk_P}$, a secret update key κ, and a disjunctive policy update P'. It outputs the updated secret proving key $\mathsf{pk_{P'}}$.

Definition 5.1 (Obliviously updateable Policy-private Zero-knowledge Authenticator)**.** *A zkAt$^+$ must additionally satisfy the following properties for policies* $\mathsf{P}, \mathsf{P}' \in \Pi$*,*

• **Update completeness:** *For every message* $M \in \mathcal{M}$ *and for every string* $\omega \in \Omega$ *such that* $\mathsf{P}'(M||\omega) = 1$*, we must have that*

$$\Pr\left[\mathsf{AuthVfy}(\mathsf{vk_P}, M, \pi) = 1 \;:\; \begin{array}{r} \mathsf{pp} \leftarrow\!\!\$\; \mathsf{Gen}(1^\lambda) \\ (\mathsf{vk_P}, \mathsf{pk_P}, \kappa) \leftarrow\!\!\$\; \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ \mathsf{pk}'_\mathsf{P} \leftarrow\!\!\$\; \mathsf{PolUpdate}(\mathsf{pk_P}, \mathsf{P}') \\ \pi \leftarrow\!\!\$\; \mathsf{AuthProve}(\mathsf{pk}'_\mathsf{P}, M, \omega) \end{array}\right] = 1$$

• **Update knowledge soundness**: *For every* PPT *adversary* $\mathcal{A}$ *there exists a* PPT *extractor* $\mathcal{E}$ *and a negligible function* $\epsilon(\cdot)$*, for all* $\lambda \in \mathbb{N}$ *such that the following probability is at most* $\epsilon(\lambda)$*,*

$$\Pr\left[\begin{array}{l} \mathsf{AuthVfy}(\mathsf{vk_P}, M^*, \pi^*) = 1 \;\wedge \\ \forall \mathsf{P}' \in Q_\mathsf{P} \;:\; \mathsf{P}'(M^*||\omega^*) \neq 1 \end{array} \;:\; \begin{array}{r} \\ \\ \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{P} \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{vk_P}, \mathsf{pk_P}, \kappa) \leftarrow\!\!\$\; \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ ((M^*, \pi^*);\; \omega^*) \leftarrow (\mathcal{A}||\mathcal{E})^{\mathcal{O}_\kappa}(\mathsf{pp}) \end{array}\right]$$

where the oracle $\mathcal{O}_\kappa(\cdot)$ *takes as input an disjunctive policy update* $\mathsf{P}^{(i)} \in \Pi$ *in its* i^{th}*-query. It adds* $\mathsf{P}^{(i)}$ *to the set* Q_P *(initially set to* $\{\mathsf{P}\}$*) and outputs the signing key* $\mathsf{pk}_\mathsf{P}^{(i)}$ *under* $\mathsf{P}^{(i)}$ *by running* $\mathsf{PolUpdate}(\mathsf{pk_P}, \kappa, \mathsf{P}^{(i)})$*.*

Remark 5.2. Since the verification key $\mathsf{vk_P}$ is independent of the policy updates, the view of the policy privacy adversary remains unaltered from that in Definition 3.1. So, policy privacy is actually implicit in the definition.

5.1 Construction

In this section, we describe our generic zkAt$^+$ construction for *disjunctive* policy updates, which we explain next.

Disjunctive policy updates. Let $\mathsf{P} \in \Pi$ be an existing policy. Then, an update $\mathsf{P}' \in \Pi$ is a disjunctive update if and only if P' is a disjunction of P with some other valid policy. Thus, for each $\mathsf{P} \in \Pi$, we can define the predicate $\mathsf{Adm_P}(\mathsf{P}') = 1 \Leftrightarrow \mathsf{P}' \in \{\mathsf{P} \vee f \;:\; \forall f \in \Pi\}$.

Barring the trivial idea of re-running the setup and distributing fresh keys under the new policy, obliviously performing *general* policy updates, which is to say non-disjunctive updates, appears to be somewhat challenging. This is because it would require a way to revoke a proving key under an older policy without also revoking the corresponding verification key. However, as we explain later, allowing the older proving key to expire via a simple tagging mechanism suffices for general policy updates with only minor generic modification to the overall construction.

At a high level, our zkAt$^+$ construction, recursively composes NIZK proofs with the "inner" proof corresponding to the policy predicate, and the "outer" proof to the *knowledge* of a valid proof to the policy predicate as well as a signature on the inner verification key for soundness. Thus a verifier simply verifies this outer proof and is convinced of the user's authenticity. Like in our Groth16-zkAt construction, our zkAt$^+$ construction also requires maintaining a sensitive secret κ that gets used whenever the policies need to be updated, so that when a new policy is created, the issuer simply issues a fresh signature on the new inner verification key (using κ). Importantly, no change is made to the outer keys so that all third parties remain unaware of the update.

Tools required. The construction below utilizes a signature scheme $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$, an inner NIZKAoK scheme $\mathsf{NIZK}_I = (\mathsf{NIZK}_I.\mathsf{Setup}, \mathsf{NIZK}_I.\mathsf{Prove}, \mathsf{NIZK}_I.\mathsf{Verify})$ which encodes the policy, and an outer NIZKAoK scheme, $\mathsf{NIZK}_O = (\mathsf{NIZK}_O.\mathsf{Setup}, \mathsf{NIZK}_O.\mathsf{Prove}, \mathsf{NIZK}_O.\mathsf{Verify})$ for the following relation

$$\mathscr{R}_O = \left\{ \begin{array}{c} x := (\mathsf{vk}_\sigma, M) \\ w := (\mathsf{crs}_I, \pi_I, \sigma) \end{array} : \begin{array}{r} \mathsf{NIZK}_I.\mathsf{Verify}(\mathsf{crs}_I, M, \pi_I) = 1 \\ \wedge\ \mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I, \sigma) = 1 \end{array} \right\}$$

• $\underline{\mathsf{Gen}(1^\lambda) \to \mathsf{pp}}$. Given the security parameter λ as input, the parameter generation algorithm generates the NIZK crs as $\mathsf{crs}_O \leftarrow\!\$\ \mathsf{NIZK}_O.\mathsf{Setup}(1^\lambda, \mathsf{C}_O)$, where C_O is the circuit encoding $\mathscr{R}_O$, and outputs $\mathsf{pp} := \mathsf{crs}_O$. This algorithm is run once and for all.
• $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}, \kappa)}$. Given the security parameter λ and the policy P as input, the setup algorithm runs the inner NIZK setup to obtain $\mathsf{crs}_I \leftarrow\!\$\ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P})$. Next, it creates the signing and verification keys for the signature scheme as $(\mathsf{vk}_\sigma, \mathsf{sk}_\sigma) \leftarrow\!\$\ \mathsf{Sig.Setup}(1^\lambda)$, and then signs crs_I to obtain $\sigma \leftarrow\!\$\ \mathsf{Sig.Sign}(\mathsf{sk}_\sigma, \mathsf{crs}_I)$. Finally, it outputs $\mathsf{vk}_\mathsf{P} := \mathsf{vk}_\sigma$, $\mathsf{pk}_\mathsf{P} := (\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$, and $\kappa := \mathsf{sk}_\sigma$.
• $\underline{\mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega) \to \pi/\bot}$. It first parses pk_P as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$ and continues only if $\mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I, \sigma) = 1$, otherwise it aborts and outputs $\bot$. It then computes the proof $\pi_I \leftarrow\!\$\ \mathsf{NIZK}_I.\mathsf{Prove}(\mathsf{crs}_I, x := M, w := \omega)$ and then outputs the final proof π computed as $\pi \leftarrow\!\$\ \mathsf{NIZK}_O.\mathsf{Prove}(\mathsf{pp}, x := (\mathsf{vk}_\sigma, M), w := (\mathsf{crs}_I, \pi_I, \sigma))$.
• $\underline{\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) \to \{0,1\}}$. Returns the output of $\mathsf{NIZK}_O.\mathsf{Verify}(\mathsf{pp}, x := (\mathsf{vk}_\mathsf{P}, M), \pi)$.
• $\underline{\mathsf{PolUpdate}(\mathsf{pk}_\mathsf{P}, \kappa, \mathsf{P}') \to \mathsf{pk}'_\mathsf{P}/\bot}$. If $\mathsf{Adm}_\mathsf{P}(\mathsf{P}') \neq 1$, it outputs $\bot$ and aborts. Otherwise, it parses pk_P as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$. Then, the update algorithm re-runs the inner NIZK setup with this input to obtain $\mathsf{crs}'_I \leftarrow\!\$\ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P}')$. Next, it signs crs'_I to obtain $\sigma' \leftarrow\!\$\ \mathsf{Sig.Sign}(\kappa, \mathsf{crs}'_I)$. Finally, it outputs $\mathsf{pk}'_\mathsf{P} := (\mathsf{crs}'_I, \mathsf{vk}_\sigma, \sigma')$.

Remark 5.3. Depending on the application scenario, the setup could be combined with the parameter generation algorithm so that both are performed once, and the rest of the protocol proceeds identically. This could be potentially useful in a situation where, for instance, an organization has an authorized list of users who can create transactions on behalf of the organization, while keeping their individual identities private. Moreover, this gives a *maximally* private authentication scheme while still demanding accountability from the users. On the other hand, if a protocol requires identities from individual users, one can perform the setup for each user and set the (hash of) vk_σ as their corresponding addresses.

Security. The main security theorem and the corresponding proof for this section are provided in Appendix D.

5.2 General Policy Updates

As we previously remarked, we can make generic modifications to the construction allows one to perform general policy-updates obliviously. We now elaborate on these ideas by giving the concrete constructions that accomplishes this, and highlight the differences from the original construction.

5.2.1 Using timestamps for general updates

Our first idea is to timestamp the policies with an expiration time so that policies past a certain expiration time are considered voided and will not be accepted by the verifiers. More concretely, we define the outer NIZKAoK scheme, NIZK_O for the following relation

$$\mathscr{R}_O^{\mathsf{time}} = \left\{ \begin{array}{l} x := (\mathsf{vk}_\sigma, M, \mathsf{t_{cur}}) \\ w := (\mathsf{crs}_I, \pi_I, \boxed{\mathsf{t_{exp}}}, \sigma) \quad : \\ \qquad\qquad\qquad \mathsf{NIZK}_I.\mathsf{Verify}(\mathsf{crs}_I, M, \pi_I) = 1 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \boxed{\wedge\ \mathsf{t_{exp}} > \mathsf{t_{cur}}} \\ \qquad\qquad \wedge\ \mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I || \boxed{\mathsf{t_{exp}}}, \sigma) = 1 \end{array} \right\}$$

The corresponding construction is as follows:

- $\underline{\mathsf{Setup}(1^\lambda, (\mathsf{P}, \mathsf{t_{exp}})) \to (\mathsf{vk_P}, \mathsf{pk_P}, \kappa)}$. Given the security parameter λ, the policy P and a timestamp $\mathsf{t_{exp}}$ as input, the setup algorithm runs the inner NIZK setup to obtain $\mathsf{crs}_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P})$. Next, it creates the signing and verification keys for the signature scheme as $(\mathsf{vk}_\sigma, \mathsf{sk}_\sigma) \leftarrow\$ \mathsf{Sig.Setup}(1^\lambda)$, and then signs crs_I to obtain $\sigma \leftarrow\$ \mathsf{Sig.Sign}(\mathsf{sk}_\sigma, \mathsf{crs}_I || \mathsf{t_{exp}})$. Finally, it outputs $\mathsf{vk_P} := \mathsf{vk}_\sigma$, $\mathsf{pk_P} := (\mathsf{crs}_I, \mathsf{vk}_\sigma, \mathsf{t_{exp}}, \sigma)$ and $\kappa := \mathsf{sk}_\sigma$.
- $\underline{\mathsf{AuthProve}(\mathsf{pk_P}, (M, \mathsf{t_{cur}}), \omega) \to \pi/\bot}$. It first parses $\mathsf{pk_P}$ as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$ and continues only if $\mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I, \sigma) = 1$, otherwise it aborts and outputs $\bot$. It then computes the proof $\pi_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Prove}(\mathsf{crs}_I, x := M, w := \omega)$ and then outputs the final proof π computed as $\pi \leftarrow\$ \mathsf{NIZK}_O.\mathsf{Prove}(\mathsf{pp}, x := (\mathsf{vk}_\sigma, M, \mathsf{t_{cur}}), w := (\mathsf{crs}_I, \pi_I, \mathsf{t_{exp}}, \sigma))$.
- $\underline{\mathsf{AuthVfy}(\mathsf{vk_P}, (M, \mathsf{t'_{cur}}, \mathsf{t_{cur}}, \varepsilon), \pi) \to \{0,1\}}$. If

$$| \mathsf{t_{cur}} - \mathsf{t'_{cur}} | \geq \varepsilon ,$$

then it outputs 0. Otherwise, it returns the output of $\mathsf{NIZK}_O.\mathsf{Verify}(\mathsf{pp}, x := (\mathsf{vk_P}, M, \mathsf{t'_{cur}}), \pi)$.
- $\underline{\mathsf{PolUpdate}(\mathsf{pk_P}, \kappa, (\mathsf{P}', \mathsf{t'_{exp}})) \to \mathsf{pk'_P}/\bot}$. If $\mathsf{Adm_P}(\mathsf{P}') \neq 1$, it outputs $\bot$ and aborts. Otherwise, it parses $\mathsf{pk_P}$ as $(\mathsf{pk}_I, \mathsf{vk}_I, \mathsf{vk}_\sigma, \sigma)$. Then, the update algorithm re-runs the inner NIZK setup with this input to obtain $\mathsf{crs}'_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P}')$. Next, it signs crs'_I to obtain $\sigma' \leftarrow\$ \mathsf{Sig.Sign}(\kappa, \mathsf{crs}'_I || \mathsf{t'_{exp}})$. Finally, it outputs $\mathsf{pk'_P} := (\mathsf{crs}'_I, \mathsf{vk}_\sigma, \mathsf{t'_{exp}}, \sigma')$.

Remark 5.4. This construction does not preclude the situation where an old proving key is still usable before it has expired. However, this may be acceptable, and even desirable, in many practical situations where, for instance, an authority may want to allow the provers some time before transitioning to a new policy.

5.2.2 Using tagged policies for general updates

An alternative to the use of timestamps is to essentially embed a tag inside the reference string of the policy, and then prove that the (private) tag is in a (public) set $\mathcal{T}$ of accepted policies for the authentication. This even circumvents the drawback of the timestamping approach, as the authority can update $\mathcal{T}$ in the public parameters at will, and publish them for the verifiers to use. More concretely, we define the outer NIZKAoK scheme, NIZK_O for the following relation

$$\mathcal{R}_O^{\mathsf{tag}} = \left\{ \begin{array}{ll} x := (\mathsf{vk}_\sigma, M, \mathcal{T}) & : \\ w := (\mathsf{crs}_I, \mathsf{tag}, \pi_I, \sigma) & \\ & \mathsf{NIZK}_I.\mathsf{Verify}(\mathsf{crs}_I, M, \pi_I) = 1 \\ & \wedge\ \mathsf{tag} \in \mathcal{T} \\ & \wedge\ \mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I || \mathsf{tag}_I, \sigma) = 1 \end{array} \right\}$$

The corresponding construction is as follows:

- $\underline{\mathsf{Setup}(1^\lambda, (\mathsf{P}, \mathsf{tag})) \to (\mathsf{vk_P}, \mathsf{pk_P}, \kappa)}$. Given the security parameter λ, the policy P and a tag $\mathsf{tag} \in \mathcal{T}$ as input, the setup algorithm runs the inner NIZK setup to obtain $\mathsf{crs}_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P})$. Next, it creates the signing and verification keys for the signature scheme as $(\mathsf{vk}_\sigma, \mathsf{sk}_\sigma) \leftarrow\$ \mathsf{Sig.Setup}(1^\lambda)$, and then signs crs_I along with tag to obtain $\sigma \leftarrow\$ \mathsf{Sig.Sign}(\mathsf{sk}_\sigma, \mathsf{crs}_I || \mathsf{tag})$. Finally, it outputs $\mathsf{vk_P} := \mathsf{vk}_\sigma$, $\mathsf{pk_P} := (\mathsf{crs}_I, \mathsf{vk}_\sigma, \mathsf{tag}, \sigma)$ and $\kappa := \mathsf{sk}_\sigma$.
- $\underline{\mathsf{AuthProve}(\mathsf{pk_P}, M, \omega) \to \pi/\bot}$. It first parses $\mathsf{pk_P}$ as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \mathsf{tag}, \sigma)$ and continues only if $\mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I || \mathsf{tag}, \sigma) = 1$, otherwise it aborts and outputs $\bot$. It then computes the proof $\pi_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Prove}(\mathsf{crs}_I, x := M, w := \omega)$ and outputs the final proof π computed as $\pi \leftarrow\$ \mathsf{NIZK}_O.\mathsf{Prove}(\mathsf{crs}_O, x := (\mathsf{vk}_\sigma, M, \mathcal{T}), w := (\mathsf{crs}_I, \pi_I, \mathsf{tag}, \sigma))$.

- $\underline{\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) \to \{0,1\}}$. It returns the output of $\mathsf{NIZK}_O.\mathsf{Verify}(\mathsf{crs}_O, x := (\mathsf{vk}_\sigma, M, \mathcal{T}), \pi)$.
- $\underline{\mathsf{PolUpdate}(\mathsf{pk}_\mathsf{P}, \kappa, (\mathsf{P}', \mathsf{tag}')) \to \mathsf{pk}'_\mathsf{P}/\bot}$. If $\mathsf{Adm}_\mathsf{P}(\mathsf{P}') \neq 1$, it outputs $\bot$ and aborts. Otherwise, it parses pk_P as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \mathsf{tag}, \sigma)$. If $\mathsf{tag}' \notin \mathcal{T}$ it outputs $\bot$ and aborts. Otherwise, the update algorithm re-runs the inner NIZK setup with this input to obtain $\mathsf{crs}'_I \leftarrow\!\!\$\ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P}')$. Next, it signs crs'_I and tag' to obtain $\sigma' \leftarrow\!\!\$\ \mathsf{Sig.Sign}(\kappa, \mathsf{crs}'_I || \mathsf{tag}')$. Finally, it outputs $\mathsf{pk}'_\mathsf{P} := (\mathsf{crs}'_I, \mathsf{vk}_\sigma, \mathsf{tag}', \sigma')$.

6 Application: Private On-chain Authentication

We now discuss how our zkAt construction can be integrated into a blockchain. First, recall that zkAt$^+$ uses existing NIZKs in a black-box manner, so its instantiation is relatively straightforward. Even our concrete Groth16-zkAt uses the standard Groth16 proving and verification algorithms. Therefore, the only major requirement for integrating our zkAt constructions is the support for on-chain NIZK verification. Fortunately, many smart-contract supporting chains like Ethereum, Aptos and Sui already support on-chain NIZK verification for Groth16 (among others), and can thus readily integrate any of our constructions. Therefore, zkAt can act as a drop-in replacement for any existing authenticator used on public blockchains including multi-signature, threshold signatures and smart-contract based authenticators.

Next, we give a practical instantiation of our zkAt. As a concrete example we will consider the policy—"*require t-out-of-n signatures*," where t, n and the n signature verification keys are all to be hidden with the zkAt. The concrete zkAt would look as follows:

- **Setup.** Create commitments $c_1, c_2, \ldots, c_n$ to the n account addresses, i.e., $c_i \leftarrow \mathsf{Commit}(\mathsf{addr}_i; r_i)$ where address addr_i is a signature verification key for the i^{th} signer. Then, create a Merkle tree digest of the n commitments, $\mathsf{root} \leftarrow \mathsf{MT.Commit}(\{c_1, \ldots, c_n\})$. Run the Groth16-zkAt setup to create a (designated-prover) NIZK proving key and (proof) verification key pair $(\mathsf{pk}_\mathsf{P}, \mathsf{vk}_\mathsf{P}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P})$ for the policy P given by the relation,

$$\left\{ \begin{array}{r} x := (\mathsf{root}, \mathsf{tx_{pb}}), \qquad\qquad\qquad\qquad \\ w := \left(\{\mathsf{addr}_i, r_i, c_i, \sigma_i\}_{i=1}^t, \mathsf{tx_{pv}}\right) \quad : \\ \forall i \neq k \in [t]\ ,\ \mathsf{addr}_i \neq \mathsf{addr}_k \\ \wedge\ c_i = \mathsf{Commit}(\mathsf{addr}_i;\ r_i) \\ \wedge\ \mathsf{Sig.Verify}(\mathsf{addr}_i, \mathsf{tx_{pb}} || \mathsf{tx_{pv}}, \sigma_i) = 1 \\ \wedge\ \mathsf{MT.Includes}(\mathsf{root}, c_i) = 1 \end{array} \right\},$$

where $\mathsf{tx_{pb}}$ (resp. $\mathsf{tx_{pv}}$) is the public (resp. private) part of the transaction. The proof verification key vk acts as the user's on-chain address.
- **Sign.** Split the transaction into public and private components, as determined by the privacy requirements. Collect signatures $\{\sigma_1, \sigma_2, \ldots, \sigma_t\}$ on the full transaction $\mathsf{tx_{pb}} || \mathsf{tx_{pv}}$ from t accounts $\{i_1, i_2, \ldots, i_t\}$. Finally, generate the authentication proof with public input $M := (\mathsf{root}, \mathsf{tx_{pb}})$ and private input $\omega := \left(\{\mathsf{addr}_i, r_i, c_i, \sigma_i\}_{i=1}^t, \mathsf{tx_{pv}}\right)$, i.e., $\pi \leftarrow\!\!\$\ \mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega)$ proving that the user has t valid signatures on the transaction from the set of n signers committed in the Merkle tree.

In particular, this design only requires implementing t signature verifications in the circuit. Assuming the use of a NIZK-friendly signature scheme which only requires a few thousand R1CS constraints per verification, this is concretely efficient (cf. § 7). Using standard signature schemes (which might be necessary if we want to use existing accounts) can lead to costlier circuits. Concretely, each secp256k1 verification requires 1.5M constraints [0xP24] and Ed25519 verification requires 2.5M constraints [Ele24]. However, proving only takes a few seconds on cloud machines[4], so these are still practical for reasonable t values seen in practice.

Remark 6.1. Successfully integrating zkAt into a blockchain, will also require some standardization effort with regards to the zkAt's public inputs so as not to unintentionally leak some information about

[4]It only takes 6s to verify Ed25519 signatures on a 16-core 32G RAM machine [Ele24].

the policy by virtue of using any specific public input. Below, we provide a brief list of the required public inputs to a zkAt. Note that a blockchain designer may decide to support all or a subset of these.

• **Transaction details ($\mathsf{tx}_{\mathsf{pb}}$):**

  – **Digest:** specifying just a transaction digest however requires parsing the transaction within a zero-knowledge proof. Transaction parsing can be simplified by carefully designing the format of a transaction, e.g., structure it in the format of a Merkle tree. We leave concrete specification for future work.

  – **Amount:** if supporting a specific type of transaction is enough, e.g., amount transfers, then exposing the transaction amount as a public input can make the signature generation process very efficient.

  – **Other fields:** similarly, other common fields of a transaction, e.g., the sender address, can be exposed as public inputs.

• **Time:** most blockchains support some notion of time. Including time allows specifying time-based policies, e.g., use a certain set of credentials before market close and another after. If the trapdoor is persisted in a safe place (e.g., cold storage), then time allows oblivious policy updates. Say we want to rotate keys once a month, then we can embed an expiry date after a month, and use the trapdoor to generate a new policy when needed.
• **Support for web2 credentials:** certain blockchains support authentication based on existing credentials issued by web2 providers, e.g., e-Passports [Int24], OpenID Connect credentials [BCJ+24]. To support these, the chains use oracles to fetch the public keys of an issuer. In this case, a Merkle root of all the authorized public keys can serve as a public input.

More authentication policy examples. By design, zkAt supports arbitrarily complex policies (i.e., policies that can be encoded as general NP statements). For this reason, it is impossible to enumerate all possible policy choices. As such, the specific policy choices will vary based on the intended application. However, to give the reader a flavor of more complex policies that motivate this work, we give another example of a policy supporting private triggers based on transaction amount—"*require t_1-out-of-n signatures for transaction amounts under k units, otherwise require t_2-out-of-n*," where t_1, t_2, n, k and the n signature verification keys are all to be hidden with the zkAt. Then, continuing our earlier example, the high-level changes to the circuit would include: adding a new hiding commitment of k, a new public input representing the actual transaction amount (in $\mathsf{tx}_{\mathsf{pb}}$), and depending on whether the transaction amount is less or more than k, check either t_1 or t_2 signatures.

7 Implementation and Evaluation

We implemented, both, our zkAt construction by instantiating with standard Groth16 (as a proxy for the Groth16-zkAt), as well as zkAt$^+$ constructions in Go using the `gnark` library [BPH+23][5] (we chose this library since it supports both Groth16 and recursive composition). We remark that we do not implement the the Groth16-zkAt construction separately, as we expect it to have an identical profile for proving and verification times to the zkAt instantiated with Groth16 since the prove and verify algorithms do not change. All our experiments were done on a laptop equipped with an Apple M3 Pro chip and we report means over 100 executions.

Policy choice. We restrict our implementations to the policy P described abstractly in the previous section as: "*require* 1-*out-of*-3 *signatures for transaction amounts under* 1000 *units, otherwise require* 2-*out-of*-3". We find that this sufficiently captures complex policy semantics not offered by, for instance, a traditional (t-out-of-n) threshold scheme while also giving a reasonable basis for comparison between the two. Of course, one can formulate more complex policies with greater number of constraints, but we note here that even as the prover time increases for more complex policies, the proof size and the verification time remain the same due to the use of zk-SNARK.

[5] GitHub: `https://github.com/Consensys/gnark`

	Signer time (ms)	**Verifier time (ms)**
zkAt for P	50.97	0.89
2-of-3 threshold	0.03	0.07

Table 2: Comparison of zkAt with threshold signatures. Signer time is equivalently the prover time in zkAt.

7.1 Evaluation of zkAt

In our proof-of-concept implementation a prover first draws an integer valued transaction amounts uniformly, creates the required number of EdDSA signatures [BDL+12] over a ZK-friendly curve and then constructs the zkAt proof with the transaction data as the public input and the signatures and their corresponding verification keys as the private input. In short, the NIZK verification circuit checks whether one or two of the signatures verify depending on the transaction amount.

Concretely, the basic zkAt was implemented over the BN254 curve and the resulting R1CS had 24,564 constraints. Table 2 compares our implementation against a 2-of-3 threshold scheme. While the signer (prover) time for zkAt is noticeably higher relative to the threshold scheme, it is still small in absolute terms. The verification times are within an order of magnitude.

7.2 Evaluation of zkAt$^+$

As before, in our simple proof-of-concept implementation a prover first draws an integer valued transaction amounts uniformly, creates the required number of EdDSA signatures [BDL+12]. It then constructs the inner proof that the necessary number of verifying signatures were obtained for the for the given transaction amount. Next, using the transaction data and its signature verification key (which can be thought of as the prover's address) as the public input, it constructs the outer proof that (i) the inner NIZK circuit accepts and; (ii) it has a verifying signature (with respect to the signature verification key corresponding to its address) on the proving key used to compute the inner proof. In our implementation, we instantiate the proof system with the Groth16 due to its compact proof size as well as support for its recursive composition inside `gnark`, although other choices of proof systems such as [GWC19, CHM+20, CBBZ23] are equally valid.

We implement the zkAt$^+$ over the SNARK-friendly 2-chain[6] of BLS12-377 inner curve [BLS03, BCG+20] and BW6-761 [BW05, EG20] outer curve which was shown to be highly efficient for Groth16 [EG20, EG22]. The inner and outer R1CSs have 24,172 and 40,474 constraints respectively. Table 3 gives the prover and verifier times for our implementation.

Inner prover time (ms)	**Preprocessing**	325.15
	Proof generation	78.55
Outer prover time (ms)		644.54
Verifier time (ms)		7.47

Table 3: Prover and verifier times for zkAt$^+$. The preprocessing time is given for each signer.

Acknowledgement

We would like to thank Foteini Baldimtsi for her feedback on the manuscript.

References

[0xP24] 0xPARC. Big integer arithmetic and secp256k1 ecc operations in circom. `https://github.com/0xPARC/circom-ecdsa`, 2024. Accessed: 2025-02-19.

[6] A 2-chain is a pair of pairing-friendly elliptic curves such that the base field of one curve is equal to the scalar field of the other. This enables efficient proof composition of up to one level [BCG+20].

[ANO+22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.

[Arg24] Argent. Smart wallet features. `https://www.argent.xyz/blog/smart-wallet-features`, 2024. Accessed: 2024-10-11.

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333, Tokyo, Japan, March 3–6, 2013. Springer Berlin Heidelberg, Germany.

[BCJ+24] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zkLogin: Privacy-preserving blockchain authentication with existing credentials. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 3182–3196, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[BCK+22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 517–550, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.

[Bit24] BitGo. Policy builder overview. `https://developers.bitgo.com/guides/policy-builder/overview`, 2024. Accessed: 2024-10-11.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267, Amalfi, Italy, September 12–13, 2003. Springer Berlin Heidelberg, Germany.

[BNO21] Dan Boneh, Wilson Nguyen, and Alex Ozdemir. Efficient functional commitments: How to commit to private functions. Cryptology ePrint Archive, Report 2021/1342, 2021.

[BPH+23] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.9.0, February 2023.

[BW05] Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

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

[DEFM19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy*, pages 106–123, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer Berlin Heidelberg, Germany.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA.

[DGL+22] Jiajun Du, Zhonghui Ge, Yu Long, Zhen Liu, Shifeng Sun, Xian Xu, and Dawu Gu. MixCT: Mixing confidential transactions from homomorphic commitment. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part III*, volume 13556 of *Lecture Notes in Computer Science*, pages 763–769, Copenhagen, Denmark, September 26–30, 2022. Springer, Cham, Switzerland.

[EG20] Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 259–279, Vienna, Austria, December 14–16, 2020. Springer, Cham, Switzerland.

[EG22] Youssef El Housni and Aurore Guillevic. Families of SNARK-friendly 2-chains of elliptic curves. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 367–396, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[Ele24] Electron Labs. Ed25519 implementation in circom. `https://github.com/Electron-Labs/ed25519-circom`, 2024. Accessed: 2024-10-11.

[Fir24] Fireblocks. Fireblocks governance and policy engine. `https://www.fireblocks.com/platforms/governance-and-policy-engine/`, 2024. Accessed: 2024-10-11.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.

[GM17] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 473–489, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[GMM$^+$22] Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 1259–1273, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[GPS06] Stephen D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, 2006.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

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

[HAB$^+$17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society.

[Int24] International Civil Aviation Organization. Basics of epassport cryptography, 2024. Accessed: 2025-02-17.

[Ita83] K. Itakura. A public-key cryptosystem suitable for digital multisignatures, 1983.

[KG20] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65, Halifax, NS, Canada (Virtual Event), October 21-23, 2020. Springer, Cham, Switzerland.

[Kos23] Kostas Kryptos Chalkias. Soft privacy-related leak in threshold eddsa wallets. `https://x.com/kostascrypto/status/1703594584100700641`, 2023.

[LAS$^+$10] Jin Li, Man Ho Au, Willy Susilo, Dongqing Xie, and Kui Ren. Attribute-based signature and its applications. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *ASIACCS 10: 5th ACM Symposium on Information, Computer and Communications Security*, pages 60–69, Beijing, China, April 13–16, 2010. ACM Press.

[LRY16] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016: 43rd International Colloquium on Automata, Languages and Programming*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14, Rome, Italy, July 11–15, 2016. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[MDM$^+$23] Easwar Vivek Mangipudi, Udit Desai, Mohsen Minaei, Mainack Mondal, and Aniket Kate. Uncovering impact of mental models towards adoption of multi-device crypto-wallets. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 3153–3167, Copenhagen, Denmark, November 26–30, 2023. ACM Press.

[Mic24] Microchain Labs. Zk session keys. `https://docs.microchain.microchainlabs.xyz/blog/second-post`, 2024. Accessed: 2025-02-21.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 245–254, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 376–392, San Francisco, CA, USA, February 14–18, 2011. Springer Berlin Heidelberg, Germany.

[RRJ+22] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2551–2564, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[Saf24] Safe Core. Safe core protocol whitepaper. `https://github.com/5afe/safe-core-protocol-specs/blob/main/whitepaper.pdf`, 2024. Accessed: 2024-10-11.

[WC23] Qin Wang and Shiping Chen. Account abstraction, analysed. In *2023 IEEE International Conference on Blockchain (Blockchain)*, pages 323–331, 2023.

A Additional Preliminaries

In this section, we provide preliminaries needed in this work.

A.1 Non-interactive linear proofs

Bitansky, et al. [BCI+13] introduced a general framework that abstracts several SNARK constructions characterized as 2-move algebraic input-oblivious linear interactive proofs, and named non-interactive linear proofs (NILP) by Groth [Gro16]. A NILP is defined by three algorithms as we describe next. The Setup algorithm takes a relation $\mathscr{R}$ as input and returns $\boldsymbol{\sigma} \in \mathbb{F}^{\mu}$ for some μ and some trapdoor information $\boldsymbol{\tau}$. The algorithm $\mathsf{ProofMatrix}$ on input $\mathscr{R}$, w and w generates a matrix $\mathbf{P} \in \mathbb{F}^{\nu \times \mu}$ for "short" proof length ν. Lastly, the algorithm Test returns vectors $\mathbf{t}_1, \dots, \mathbf{t}_\eta \in \mathbb{F}^{\mu+\nu}$. These algorithms further define the prover and verifier algorithms in an implicit way as:

Definition A.1 (Non-interactive linear proofs)**.**

- $\mathsf{Prove}(\mathscr{R}, \boldsymbol{\sigma}, x, w) \to \boldsymbol{\pi}$. *Obtains* $\mathbf{P} \leftarrow\!\$\ \mathsf{ProofMatrix}(\mathscr{R}, x, w)$ *and outputs* $\boldsymbol{\pi} := \mathbf{P}\boldsymbol{\sigma}$
- $\mathsf{Verify}(\mathscr{R}, \boldsymbol{\sigma}, x, \boldsymbol{\pi})$. *Obtains* $(\mathbf{t}_1 \dots, \mathbf{t}_\eta) \leftarrow\!\$\ \mathsf{Test}(\mathscr{R}, x)$ *and returns the outcome of*

$$\bigwedge_{i=1}^{\eta} \left(\mathbf{t}_i \begin{pmatrix} \boldsymbol{\sigma} \\ \boldsymbol{\pi} \end{pmatrix} = 0 \right)$$

Let $\mathsf{G}_{\mathscr{R}}$ *be a relation generator that, given the security parameter* λ*, returns a polynomial-time decidable binary relation* $\mathscr{R}$*, and let* $\mathscr{R}_\lambda$ *be the set of all such possible relations output by* $\mathsf{G}_{\mathscr{R}}$*. A NILP must satisfy the following properties:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, for every* $\mathscr{R} \in \mathscr{R}_\lambda$ *and every crs computed as* $\boldsymbol{\sigma} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathscr{R})$*, any instance and witness pair* $(x, w) \in \mathscr{R}$*,*

$$\Pr\left[\mathsf{Verify}(\boldsymbol{\sigma}, x, \boldsymbol{\pi}) = 1 \ :\ \boldsymbol{\pi} \leftarrow\!\$\ \mathsf{Prove}(\boldsymbol{\sigma}, x, w)\right] = 1.$$

- **Zero-knowledge:** *There exists a* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *for every adversary* $\mathcal{A}$ *and such that there is a negligible function* $\epsilon(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every* $(x, w) \in \mathscr{R}$ *the following probability is at most* $\epsilon(\lambda)$*,*

$$\left| \Pr\left[\begin{array}{r} \mathcal{A}(\boldsymbol{\sigma}, x, \mathbf{P}) = 1 : \hspace{6em} \\ \mathcal{R} \leftarrow\!\$\ \mathsf{G}_{\mathcal{R}}(1^{\lambda}) \\ \boldsymbol{\sigma} \leftarrow\!\$\ \mathsf{Setup}(1^{\lambda}, \mathcal{R}) \\ \mathbf{P} \leftarrow\!\$\ \mathsf{Prove}(\boldsymbol{\sigma}, x, w) \end{array}\right] - \Pr\left[\begin{array}{r} \mathcal{A}(\boldsymbol{\sigma}, x, \mathbf{P}) = 1 : \hspace{6em} \\ \mathcal{R} \leftarrow\!\$\ \mathsf{G}_{\mathcal{R}}(1^{\lambda}) \\ (\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \mathcal{S}_1(1^{\lambda}, \mathcal{R}) \\ \mathbf{P} \leftarrow \mathcal{S}_2(\boldsymbol{\tau}, x) \end{array}\right] \right|$$

- **Knowledge soundness (against affine prover strategies):** *For every* PPT *adversary $\mathcal{A}$ there exists a* PPT *extractor $\mathcal{E}$ and a negligible function $\epsilon(\cdot)$, such that for all $\lambda \in \mathbb{N}$ the following probability is at most $\epsilon(\lambda)$,*

$$\Pr\left[\begin{array}{r} \begin{array}{l} \mathsf{Verify}(\boldsymbol{\sigma}, x^*, \mathbf{P}^*\boldsymbol{\sigma}) = 1 \wedge \\ (x^*, w^*) \notin \mathcal{R} \ \wedge \ \mathbf{P}^* \in \mathbb{F}^{\nu \times \mu} \end{array} : \hspace{10em} \\ \mathcal{R} \leftarrow\!\$\ \mathsf{G}_{\mathcal{R}}(1^{\lambda}) \\ \boldsymbol{\sigma} \leftarrow\!\$\ \mathsf{Setup}(1^{\lambda}, \mathcal{R}) \\ ((x^*, \mathbf{P}^*);\ w^*) \leftarrow (\mathcal{A}||\mathcal{E})(\mathcal{R}, \boldsymbol{\sigma}) \end{array}\right]$$

Split NILP. For pairings of Type 3, and working in the exponent, one must split the NILP by the operating group. Thus, a split NILP reference string $\boldsymbol{\sigma}$ (similarly for the proof $\boldsymbol{\pi}$) is comprised of two vectors $\boldsymbol{\sigma}_1 \in \mathbb{F}^{\mu_1}$ and $\boldsymbol{\sigma}_2 \in \mathbb{F}^{\mu_2}$ corresponding to groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. More concretely, the $\mathsf{ProofMatrix}$ algorithm generates matrices $\mathbf{P}_1 \in \mathbb{F}_1^{\mu} \times \nu_1$ and $\mathbf{P}_2 \in \mathbb{F}_2^{\mu} \times \nu_2$, which then allows the prover to compute $\boldsymbol{\pi}_1 := \mathbf{P}_1\boldsymbol{\sigma}_1$ (similarly for $\boldsymbol{\pi}_2$); and the Test algorithm returns matrices $\mathbf{T}_1, \ldots, \mathbf{T}_\eta \in \mathbb{F}^{(\mu_1+\nu_1)\times(\mu_2+\nu_2)}$ which allows the verifier to compute

$$\bigwedge_{i=1}^{\eta} \left(\begin{pmatrix} \boldsymbol{\sigma}_1 \\ \boldsymbol{\pi}_1 \end{pmatrix} \cdot \mathbf{T}_i \begin{pmatrix} \boldsymbol{\sigma}_2 \\ \boldsymbol{\pi}_2 \end{pmatrix} = 0 \right)$$

Disclosure-freeness and knowledge soundness (in the generic group model). Groth [Gro16] defined *disclosure-free* NILP, which helps formalize the notion that the reference string reveals no information to a prover that would allow it to choose $\mathbf{P}$ maliciously.

Definition A.2 (Disclosure-free NILP)**.** *A split NILP is disclosure free if for every adversary $\mathcal{A}$ and such that there is a negligible function $\epsilon(\cdot)$ such that for every $\lambda \in \mathbb{N}$ the following probability is at least $1 - \epsilon(\lambda)$,*

$$\Pr\left[\begin{array}{r} \boldsymbol{\sigma}_1 \cdot \mathbf{T}^*\boldsymbol{\sigma}_2 = 0 \Leftrightarrow \boldsymbol{\sigma}'_1 \cdot \mathbf{T}^*\boldsymbol{\sigma}'_2 = 0 \quad : \hspace{6em} \\ \mathcal{R} \leftarrow\!\$\ \mathsf{G}_{\mathcal{R}}(1^{\lambda}) \\ \mathbf{T}^* \leftarrow \mathcal{A}(\mathcal{R}) \\ (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2), (\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2) \leftarrow\!\$\ \mathsf{Setup}(\mathcal{R}) \end{array}\right]$$

In words, the above definition simply states that with all but negligible probability, any test deviced by the adversary has identical outcomes independently of the reference string. In particular, this allows one to design a pairing-based NIZK by compiling the split NILP (in the exponent). The reader may refer to [Gro16] for further details concerning this compiler. For our purposes, we will require the following lemma regarding the resulting NIZK argument:

Lemma A.3 ([Gro16, Lemma 1])**.** *The pairing-based protocol obtained from compiling the split NILP is a non-interactive argument with perfect completeness and statistical knowledge soundness against generic adversaries that perform polynomial number of generic group operations. Furthermore, it is perfect zero-knowledge if the split NILP is perfect zero-knowledge.*

B Proofs for Section 3

Theorem B.1 (Policy privacy)**.** *Assuming the DP-NIZK scheme* NIZK *with equivocable verification keys satisfies zero-knowledge, construction 3 is policy-private.*

Observe that aside from the witness, the proving algorithm also takes the proving key as input, which in turn carries some information about the policy. Given this, our high level strategy is to simulate the NIZK proof so that it is generated independently of the witness and the proving key, which essentially guarantees that there is no leakage on the policy.

Proof. The proof proceeds through a sequence of hybrids.

$\mathbf{H}_0$: This is the vk-equivocation experiment.

$\mathbf{H}_1$: This is the same as hybrid $\mathbf{H}_0$ except that instead of running Prove, the challenger runs the simulator $\mathcal{S}$ for NIZK without the witness to generate π.

Notice that, by zero-knowledge of NIZK, hybrid $\mathbf{H}_1$ is indistinguishable from hybrid $\mathbf{H}_0$. Moreover, by definition of zero-knowledge, the simulated proof is generated *independently* of the policy relation[7]. We now show that vk_P is independent of the choice of the policy. In order to do so, we give a PPT reduction $\mathcal{B}_{\mathsf{Equiv}}$ that, given a policy-privacy attacker $\mathcal{A}$, breaks the vk-equivocation of NIZK with respect to hybrid $\mathbf{H}_1$. In particular, $\mathcal{B}_{\mathsf{Equiv}}$ does the following:

- On receiving policies P_0, P_1 from $\mathcal{A}$, the reduction forwards them to the challenger $\mathcal{C}_{\mathsf{Equiv}}$ and receives the corresponding verification key vk_b which it forwards as the $\mathsf{vk}_{\mathsf{P}_b}$ to $\mathcal{A}$.
- When $\mathcal{A}$ outputs $(M^*, \omega_0^*, \omega_1^*)$, the reduction forwards it to $\mathcal{C}_{\mathsf{Equiv}}$ and receives a (simulated) proof π that it sends back to $\mathcal{A}$.
- Finally, it outputs whatever $\mathcal{A}$ outputs.

In doing so, $\mathcal{B}_{\mathsf{Equiv}}$ wins with the same advantage as that of $\mathcal{A}$. Thus, assuming it has vk-equivocation, construction 3 must be policy-private. □

C Proofs for Section 4

Lemma 4.1. *$T(X)$ divides $\sum_{i=0}^m a_i\tilde{U}_i(X) \cdot \sum_{i=0}^m a_i\tilde{V}_i(X) - \sum_{i=0}^m a_i\tilde{W}_i(X)$ if and only if $(a_1, \ldots, a_m)$ is a satisfying assignment for $\mathcal{R}$.*

Proof. In the forward direction, notice that $T(X) \mid \sum_{i=0}^m a_i\tilde{U}_i(X) \cdot \sum_{i=0}^m a_i\tilde{V}_i(X) - \sum_{i=0}^m a_i\tilde{W}_i(X)$ implies that $\sum_{i=0}^m a_i\tilde{U}_i(r_j) \cdot \sum_{i=0}^m a_i\tilde{V}_i(r_j) = \sum_{i=0}^m a_i\tilde{W}_i(r_j)$ for every $j \in [n]$. It follows that in the corresponding circuit, the j^{th} gate operation $a_j^L \circ a_j^R = a_j^O$ for all gates j. Thus, $(a_1, \ldots, a_m)$ is a satisfying assignment for $\mathcal{R}$.

In the reverse direction, notice that if $(a_1, \ldots, a_m)$ is a satisfying assignment then for any gate $j \in [n]$ we have by construction that

$$\begin{aligned}\sum_{i=0}^m a_i\tilde{U}_i(r_j) \cdot \sum_{i=0}^m a_i\tilde{V}_i(r_j) &= \sum_{i=0}^m a_iU_i(r_j) \cdot \sum_{i=0}^m a_iV_i(r_j) \\ &= \sum_{i=0}^m a_iW_i(r_j) \;=\; \sum_{i=0}^m a_i\tilde{W}_i(r_j)\end{aligned}$$

So, we must have $(X - r_j) \mid \sum_{i=0}^m a_i\tilde{U}_i(X) \cdot \sum_{i=0}^m a_i\tilde{V}_i(X) - \sum_{i=0}^m a_i\tilde{W}_i(X)$, but the construction is general so this must hold for all $j \in [n]$. Consequently, we have $T(X) \mid \sum_{i=0}^m a_i\tilde{U}_i(X) \cdot \sum_{i=0}^m a_i\tilde{V}_i(X) - \sum_{i=0}^m a_i\tilde{W}_i(X)$. □

Theorem C.1 (vk-equivocation)**.** *The construction with respect to the modified setup algorithm described above satisfies vk-equivocation.*

Proof. The proof proceeds through a sequence of hybrids.

[7] We encourage the reader to look at the simulator described in the proof of Theorem C.1 for a concrete example of this.

$\mathbf{H}_0$: This is the vk-equivocation experiment.

$\mathbf{H}_1$: This is the same as hybrid $\mathbf{H}_0$ except that instead of running Prove, we will run the simulator $\mathcal{S}$ for NIZK without the witness to first generate, pk, and then vk and π. More precisely, given the trapdoor, the simulator simply sets $C \in \mathbb{Z}_p$ such that

$$[C]_1 \cdot [\delta]_2 = [A]_1 \cdot [B]_2 - [\alpha]_1 \cdot [\beta]_2 - \left(\sum_{i=0}^{\ell} a_i [\chi_i]_1 \right) \cdot [\gamma]_2 \ ,$$

for uniformly random $A, B \in \mathbb{Z}_p$ (this is allowed by the randomness of r and s), and outputs $\pi := ([A]_1, [B]_2, [C]_1)$ as the simulated proof. Thus, the real and simulated proofs have identical probability distributions and so hybrid $\mathbf{H}_1$ is indistinguishable from hybrid $\mathbf{H}_0$.

More importantly, the simulated proof is comprised of uniformly random group elements $[A]_1, [B]_2$, such that they uniquely determine $[C]_1$ through the verification equation. Consequently, the simulated proof itself is independent of the policy.

Now, without loss of generality (due to symmetry), suppose $b = 0$, then for every policy $\mathsf{P} \in \Pi$ and any fixed choice of $(\alpha, \beta, \gamma, \delta, x)$ there exists a choice for $(\{y_{U,i}\}_{i=0}^{m}, \{y_{V,i}\}_{i=0}^{m}, \{y_{W,i}\}_{i=0}^{m})$ such that $\mathsf{vk}_{\mathsf{P}} = \mathsf{vk}_{\mathsf{P}_b}$. In other words under the modified setup, the verification key information theoretically hides the underlying relation. Thus, in particular, there is a choice of $y_{P,i}$'s (symbol $P \in \{U, V, W\}$) such that $\mathsf{vk}_{\mathsf{P}_0} = \mathsf{vk}_{\mathsf{P}_1}$. This completes the proof. $\square$

Theorem 1 (Knowledge soundness)**.** *The construction with respect to the modified setup algorithm described above is knowledge sound.*

Our proof is in two steps. First, we will argue that the adversary's knowledge of $\tilde{\mathscr{R}}$ (having already seen $\mathscr{R}$) reveals no additional information about $\tilde{\tau}$. Given this, we then show that the scheme above has statistical knowledge soundness against adversaries that only use a polynomial number of *generic* bilinear group operations whence, from Lemma A.3, knowledge soundness will follow.

Proof. The former follows from a straightforward information theoretic argument. Notice that the only difference in $\mathscr{R}$ and $\tilde{\mathscr{R}}$ is the collection of polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^{m}$, and $\left\{\tilde{U}_i(X), \tilde{V}_i(X), \tilde{W}_i(X)\right\}_{i=0}^{m}$ respectively such that for each symbol $S \in \{U, V, W\}$, each $i \in [0, m]$, polynomials $S_i(X)$ and $\tilde{S}_i(X)$ agree on point r_j for every $j \in [n]$. Moreover, the remaining evaluation point x for every $\tilde{S}_i(X)$ is random, and evaluates to a value $y_{S,i}$ chosen uniformly and independently. Since neither x nor any of the $y_{S,i}$ are known to the adversary, the probability that the adversary learns any $y_{S,i}$ is just $1/p$. Thus, with all but negligible probability, no *additional* information is revealed to the adversary.

So we can just look at $\tilde{\mathscr{R}}$ for the latter part of the proof and make the observation that the construction above already encodes a split NILP (cf. Appendix A). Thus due to Lemma A.3, we need only show disclosure-freeness of the reference string (ie., the discrete-log of pk and vk elements). To that end, we have that the split reference string $(\tilde{\boldsymbol{\sigma}}_1, \tilde{\boldsymbol{\sigma}}_2)$ is an evaluation of polynomials in α, β, etc. So, a test of the form $\tilde{\boldsymbol{\sigma}}_1 \cdot \mathbf{T}^* \tilde{\boldsymbol{\sigma}}_2$ for some adversarially chosen test matrix $\mathbf{T}^*$ evaluates to zero if either:

(i) It is a non-zero Laurent polynomial that evaluates to zero for the specific choice of input variables.

(ii) The formal multi-variate Laurent polynomial corresponding to the test is identically zero.

Recall the Schwartz-Zippel lemma, which states that a non-zero multivariate polynoimal over $\mathbb{F}_p$ of total degree d evaluates to zero on a uniformly random point with probability at most $d/(p-1)$. Thus we can argue by the Schwartz-Zippel lemma that, in the former case, the probability that for uniformly chosen evaluation points the non-zero polynomial evaluates to zero is negligible for polynomials of degree bounded by $\mathsf{poly}(\lambda)$. Moreover in the latter case we could simply replace $\tilde{\boldsymbol{\sigma}}_1$ and $\tilde{\boldsymbol{\sigma}}_2$ by any other $\tilde{\boldsymbol{\sigma}}'_1$ and $\tilde{\boldsymbol{\sigma}}'_2$ such that also $\tilde{\boldsymbol{\sigma}}'_1 \cdot \mathbf{T}^* \tilde{\boldsymbol{\sigma}}'_2 = 0$, which was to be shown. $\square$

D Proofs for Section 5

Theorem D.1 (Update knowledge soundness)**.** *Assuming NIZK schemes* NIZK_I *and* NIZK_O *are knowledge sound, and the signature scheme* Sig *is existentially unforgeable under chosen messages, construction 5 is update knowledge sound for disjunctive updates.*

Proof. We describe the extractor $\mathcal{E}$ and show that it is able to extract a valid witness with non-negligible probability. In particular, on input pp and π^*, the extractor $\mathcal{E}$ does the following:

- It extracts a witness from π^* as $(\mathsf{crs}_I^*, \pi_I^*, \sigma^*) \leftarrow \mathcal{E}_O(\pi^*)$.
- Next, if $\mathsf{crs}_I^* = \mathsf{crs}_I^{(i)}$ for some $i \in [0, |Q_\mathsf{P}|]$, it extracts a witness from π_I^* as $\omega^* \leftarrow \mathcal{E}_I(\pi_I^*)$ and outputs it.

Let $\nu(\lambda)$ be the probability,

$$\Pr\left[\forall i \in [0, |Q_\mathsf{P}|] \ : \ \mathsf{crs}_I^* \neq \mathsf{crs}_I^{(i)}\right] \quad .$$

Also let $\mathsf{Adv}^{O}_{\mathsf{KSnd},\mathcal{A}}(\lambda)$ be an adversary's advantage in the NIZKAoK knowledge soundness experiment for NIZK_O (similarly, for NIZK_I). Then, the probability that $\mathcal{E}$ extracts successfully is

$$(1 - \mathsf{Adv}^{O}_{\mathsf{KSnd},\mathcal{A}}(\lambda)) \cdot (1 - \mathsf{Adv}^{I}_{\mathsf{KSnd},\mathcal{A}}(\lambda)) \cdot (1 - \nu(\lambda))$$

Which is at least $(1 - \epsilon(\lambda))$, for some negligible function $\epsilon(\cdot)$ if $\nu(\lambda)$ is negligible. Now suppose, for contradiction, that $\nu(\lambda)$ is non-negligible then we give a PPT reduction $\mathcal{B}_{\mathsf{EUnf}}$ that, given $\mathcal{A}$, breaks the existential unforgeability of Sig (under chosen messages). In particular, $\mathcal{B}_{\mathsf{EUnf}}$ does the following:

- On receiving the policy P from $\mathcal{A}$, the reduction runs the setup honestly, except that instead of signing the crs_I by itself, it the EUnf challenger $\mathcal{C}_{\mathsf{EUnf}}$ for the signature σ.
- On the ith update query, it runs the policy update algorithm honestly, except that instead of signing the $\mathsf{crs}_I^{(i)}$ by itself, it the EUnf challenger $\mathcal{C}_{\mathsf{EUnf}}$ for the signature $\sigma^{(i)}$.
- Finally, when $\mathcal{A}$ outputs (M^*, π^*), the reduction uses the secret trapdoor to extract a witness from π^* as $(\mathsf{crs}_I^*, \pi_I^*, \sigma^*) \leftarrow \mathcal{E}_O(\mathsf{pp}, \pi^*)$ and outputs $(\mathsf{crs}_I^*, \sigma^*)$ as its forgery.

Since, by assumption, $\mathsf{crs}_I^* \neq \mathsf{crs}_I^{(i)}$ for any $i \in [0, |Q_\mathsf{P}|]$, we must have that $(\mathsf{crs}_I^*, \sigma^*)$ is a valid forgery. Thus, $\mathcal{B}_{\mathsf{EUnf}}$ wins with probability

$$\left(1 - \mathsf{Adv}^{O}_{\mathsf{KSnd},\mathcal{A}}(\lambda)\right) \cdot \nu(\lambda) \quad .$$

It therefore follows that $\nu(\lambda)$ is negligible, and consequently, construction 5 is update knowledge sound for disjunctive updates. □

Theorem D.2 (Zero-knowledge)**.** *Assuming the NIZK scheme* NIZK_I *has perfect zero-knowledge, construction 5 is perfectly zero knowledge.*

Proof. The proof proceeds through a sequence of hybrids.

$\mathbf{H}_0$: This is the real zero-knowledge experiment for OP-zkAt.

$\mathbf{H}_1$: This is the same as hybrid $\mathbf{H}_0$ except that instead of running $\mathsf{AuthProve}$ as is, the challenger first runs the simulator $\mathcal{S}_I$ for NIZK_I without ω to generate (crs_I and) π_I, and then proceed as before.

Notice that hybrid $\mathbf{H}_1$ is just the simulated zero-knowledge experiment for OP-zkAt. By perfect zero-knowledge of NIZK_I, hybrid $\mathbf{H}_1$ is perfectly indistinguishable from hybrid $\mathbf{H}_0$, and consequently, construction 5 is perfectly zero knowledge. □

Theorem D.3 (Policy privacy)**.** *Assuming the NIZK scheme* NIZK_O *has computational zero-knowledge, Construction 5 is policy private.*

Proof. The proof proceeds through a sequence of hybrids.

$\mathbf{H}_0$: This is the real zero-knowledge experiment for OP-zkAt.

$\mathbf{H}_1$: This is the same as hybrid $\mathbf{H}_0$ except that instead of running $\mathsf{AuthProve}$, the challenger runs the simulator $\mathcal{S}_O$ for NIZK_O without the witness to generate (crs_O and) π.

By computational zero-knowledge of NIZK_O, hybrid $\mathbf{H}_1$ is computationally indistinguishable from hybrid $\mathbf{H}_0$. Furthermore, notice that vk_P is independent of the choice of the policy. It therefore follows that construction 5 is policy private. □