zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure

Michael Rosenberg[*1], Jacob White[†2], Christina Garman[2], and Ian Miers[1]

[1]University of Maryland, {micro, imiers}@umd.edu
[2]Purdue University, {white570, clg}@purdue.edu

July 19, 2023

Abstract

Frequently, users on the web need to show that they are, for example, not a robot, old enough to access an age restricted video, or eligible to download an ebook from their local public library without being tracked. Anonymous credentials were developed to address these concerns. However, existing schemes do not handle the realities of deployment or the complexities of real-world identity. Instead, they implicitly make assumptions such as there being an issuing authority for anonymous credentials that, for real applications, requires the local department of motor vehicles to issue sophisticated cryptographic tokens to show users are over 18. In reality, there are multiple trust sources for a given identity attribute, their credentials have distinctively different formats, and many, if not all, issuers are unwilling to adopt new protocols.

We present and build `zk-creds`, a protocol that uses general-purpose zero-knowledge proofs to 1) remove the need for credential issuers to hold signing keys: credentials can be issued to a bulletin board instantiated as a transparency log, Byzantine system, or even a blockchain; 2) convert existing identity documents into anonymous credentials without modifying documents or coordinating with their issuing authority; 3) allow for flexible, composable, and complex identity statements over multiple credentials. Concretely, identity assertions using `zk-creds` take less than 150ms in a real-world scenario of using a passport to anonymously access age-restricted videos.

[*]ORCID 0000-0001-9784-125X
[†]ORCID 0000-0002-6850-2133

1 Introduction

Privacy-preserving identification is an apparent contradiction in terms: one cannot both wish to simultaneously identify themselves and stay private. But this is increasingly necessary on today's internet. For example, Australia, the EU, and the UK age-restrict access to some video content, requiring identification via a credit card or photo of an official ID to access it [Goob, Gooa]. The tracking and data exposure risks raised by such requirements can be eliminated with privacy-preserving cryptography: anonymous credentials allow a user to assert that they meet some access criteria, e.g., are over 18, without revealing anything else about themselves, linking their viewing habits to their identity, or even linking distinct video views together. Beyond this narrow application, anonymous credentials could be extended to complex identity statements—for example, checking residency for accessing online library resources or petitioning local elected representatives[1]—and the composition of credentials such as the pairing of a vaccine card with a photo ID.

While the subject of extensive academic work [Cha85, CL03, CL04, BCKL08, BL13, GGM14, CDHK15, SAB+19], anonymous credentials have thus far seen little deployment.[2] In large part, this is because most existing systems are designed with a number of assumptions about identity that, while suitable for advancing a body of cryptographic knowledge, produce designs that can be difficult to actually deploy in real-world identity systems.

Existing anonymous credential schemes make, at a minimum, some subset of the following simplifying assumptions: there is a single issuer for a given identity property (e.g., date of birth); when there are multiple issuers for a property, the property formats are compatible; there exist reputable authorities that are able and (more importantly) willing to be responsible for holding signing keys, verifying identity properties, and running sophisticated cryptographic protocols for issuing anonymous credentials; all attribute formats needed for a credential are known in advance; and the set of authorities for a given identity attribute or credential can be enumerated at the time one instantiates the system.

In this paper, we build `zk-creds`, a flexible, issuer-agnostic anonymous credential toolkit for complex identity statements. The general-purpose proving functionality supported by zkSNARKs gives us the flexibility to address most of these challenges. From this approach, we automatically gain support for ad-hoc composition of credentials and access criteria, and issuance by (threshold) signatures. In many cases, we can even remove the need for trusted anonymous credential issuers entirely by instantiating a *bulletin board* to track issued

[1] New York State provides such a platform with no privacy guarantees [New].

[2] Notable exceptions for human uses of credentials are limited trials of CL-sigs with Idemix [CV02] and, although it is not a full-fledged anonymous credential scheme, Privacy Pass [DGS+18]. Intel also makes use of DAA [BCC04] for device attestation, and a MAC variant of anonymous credentials is also being used for private groups in Signal [CPZ20].

credentials, no longer requiring issuers to hold signing keys or other secrets. Concretely, this work contributes the design and implementation of a toolkit for flexible privacy-preserving credentials, and builds two example applications.

1.1 Past work and real-world limitations

Several approaches have tried to address the limitations of anonymous credentials, focusing primarily on the problems of finding and trusting issuers.

Distributing issuance via multiple issuers. To reduce the trust needed in issuers, schemes have explored threshold issuance [SAB+19] and support for multiple issuers [CL01]. While this improves the situation if there are multiple willing issuers, it does not address the potential scarcity of issuers who are willing or able to deploy novel (or any) cryptography. Nor does it provide a means to reconcile the differing identity document formats or use cases multiple issuers would have.

Decentralizing issuance by removing signing keys. In Decentralized Anonymous Credentials [GGM14], credentials are maintained in some form of transparency log which can either be centralized and audited, distributed across cooperating parties, or operated in a decentralized fashion by a Byzantine system or blockchain. While this approach removes one obstacle to credential issuance by avoiding signing keys, the concrete protocol has performance and operational limitations. For example, the protocol requires that all clients have the full list of issued credentials, and does not address any of the other complexities of real use cases.

The messy reality of identity claims. We now return to our initial example: an anonymous credential to allow access to age-restricted videos and prevent tracking of browsing habits. In theory, whichever authority issues identity cards in a country can also issue anonymous digital credentials to everyone of age. But in practice, a number of problems arise when attempting to deploy such a scheme with existing anonymous credentials.

First, there is not a single source of identity documents (e.g., the US has 50+ drivers license issuers) and few might wish to participate due to the burden of deploying new technology. Fewer still can be trusted to secure the requisite signing keys for issuing credentials.

Second, requirements will change. What started as a token for being over 18 will need to support other age checks—under 12, over 21, over 65—necessitating more complex credentials, access criteria, and potentially credential revocation and reissuance.

Third, each ID issuer will, by default, form its own anonymity set. Even for "multi-authority" schemes designed to avoid this, differences in data fields can distinguish populations:[3] a foreign diplomat accessing age-restricted content in their host country may be distinguished from a resident using a local ID.

[3]Consider something as simple as date formats: Japanese Drivers Licenses give birth year

Fourth, new identity documents need to be integrated as they emerge to avoid access equity issues, and these documents may have differing formats. For example, many cities now issue IDs in part for undocumented residents [IDN].

Finally, even for something as conceptually simple as "of age," identity statements are not necessarily simple: in the event age limits differ between jurisdictions, a video platform needs to check where the viewer is located, and IP geolocation may be insufficient (e.g, in the case of Tor or a VPN). Credentials can directly encode a home address but, even for physical credentials, this does not work in practice: people move and do not update their IDs, and as a result need to provide alternative proofs of address. Supporting this privately requires composing credentials for, e.g., age and residency.

Minimizing trust when issuing credentials. Current (even non)-anonymous credential protocols assume the same party verifies claimed identity properties and signs cryptographic credentials. This requires finding a single party who is trusted for two different (and not necessarily related) tasks: one who is both capable of verifying identity attributes and competent to manage signing keys. The linking of these two roles is often unnecessary and complicates deployment.

First, many uses of anonymous credentials do not use identity attributes which must be verified by a trusted party to issue a credential. Looking ahead, we describe an issuer-less Privacy Pass-like construction [DGS+18] where Sybil-resistant anonymous tokens are issued by making a blockchain payment. This has no trusted parties—neither for verifying that the user is not a Sybil, nor for signing a credential.

Second, even when we must trust some entity to verify identity attributes (e.g., a passport issuer for a user's date of birth), it is not necessary to trust an additional party to hold key material for a novel cryptographic scheme. Looking ahead, we offer the minimal trust assumptions in many such cases by replacing signing with a list maintained by a centralized party or a distributed bulletin board.

Third, even where there is a trusted party who both verifies identity attributes and issues credentials, trusting a party to maintain a list is safer than trusting them to secure signing keys. In existing anonymous credential schemes, compromise of issuing keys is frequently undetectable and rollback requires rekeying and reissuing. With issuance via a list, compromise is detectable and easily reversible. This is true even if the list is maintained by a single party.

All of the aforementioned issues can be addressed by a scheme that is flexible, dynamically adaptable to new use cases post-deployment, minimizes the need to find new trusted parties, and can support complex access criteria

relative to eras that correspond to the reign of the emperor. However, they use the Gregorian calendar for months and days.

that are agnostic to the issuer or credential format.

1.2 Our contribution

We introduce `zk-creds`, a toolkit for privacy-preserving authentication protocols and anonymous credentials that offers flexible identity assertions and does not need trusted issuers. A key contribution of `zk-creds` over previous works is the usage of general-purpose zero-knowledge proofs rather than bespoke proof systems over blind signatures.

The switch to general-purpose zero-knowledge proofs as the basis for anonymous credentials, instead of blind signatures, is a paradigm shift: rather than imagining a subset of use cases and designing custom protocols for each while balancing cryptographic tradeoffs, `zk-creds` gets full privacy and full expressivity even after a protocol is designed and deployed. A single scheme, built with `zk-creds`, is adaptable to shifting requirements without requiring the development of new custom cryptographic protocols. Moreover, application-specific logic can be defined and modified in simple programming languages via a number of publicly available tools [MTBI+22, Ar22, Bow17a] with the instantiation of the scheme handled by the compiler.

General-purpose zero-knowledge proofs enable `zk-creds` to support flexible and composable access criteria. `zk-creds` not only allows users to privately show that their credential(s) meet some arbitrary access criteria check, but it also allows these criteria to be defined at any time (even after system setup or credential issuance), by any party, and composed dynamically as *gadgets*. This flexibility allows `zk-creds` to meet the reality of real-world authentication mechanisms: requirements can dynamically change at any time, as can use cases and even identity issuers.

The second major contribution of `zk-creds` is its support of existing government identity infrastructure without modification or collaboration. Using general-purpose zero-knowledge proofs, we can convert the digital (non-anonymous) identity information that is increasingly included in national identity cards and passports into anonymous credentials. This *zk-supporting-documentation* allows us to provide a digital analogue of walking into a US Department of Motor Vehicles and presenting existing identity documents to get a driver's license, *without* exposing any information to the issuer. We implement an end-to-end example of this paradigm, using a zero-knowledge proof over the data in unmodified NFC-enabled US passports to create credentials for accessing age-restricted videos.

As a third contribution, `zk-creds` supports publicly verifiable credentials. Because the issued credential list can be maintained by a public system (i.e., a transparency log or blockchain) and each credential can include zk-supporting-documentation justifying its issuance, the set of all issued credentials is publicly auditable. This is not possible when credentials can be surreptitiously issued

via signing keys.[4] As such, we need only trust the issuer to add credentials (and their supporting documentation) to the list, and any compromise or malfeasance is detectable and reversible.

To summarize, in this paper we design, build, and benchmark `zk-creds` which:

- drastically improves performance over existing decentralized schemes via reusable proofs where ShowCred takes < 150ms;
- supports existing physical identity documents (e.g., passports) without modification via zk-supporting-documentation;
- provides support for flexible and composable gadgets that can be combined to express complex access criteria checks even after system setup;
- allows for public auditability of issued credentials, without harming anonymity;
- provides (of independent interest), blind Groth16, a novel mechanism for privately linking together multiple zero-knowledge proofs in a way that enables proof rerandomization and reuse; and
- includes a full application for age-restricted video access with cloning resistance, using existing passports for issuance.

Non-goals: On-chain verification of credentials. This work constructs flexible credentials that can be *issued* without a central party holding a signing key (although we also support signature-based issuance). This should not be confused with a different area of both industrial and academic work (see e.g., [RPX+22]), which considers *verification* of existing (i.e., centrally issued) anonymous credentials by a smart contract. The question for on-chain verification of anonymous credentials is not how to remove centralized issuance, but simply how to minimize the cost of verification given the extreme cost of smart contract execution. Reducing verification costs is typically done by batching verification inside a zero-knowledge proof[5] and is generically applicable to any anonymous credential scheme, including the ones proposed here.

2 Overview

`zk-creds` is a system for issuing credentials to users and privately showing that a credential meets access criteria.

2.1 Example application and credential lifecycle

A *credential* is a commitment to a set of attributes (e.g., name, date of birth, etc.). A credential is *issued* (see Figure 1) when it is made a leaf in a Merkle

[4] We note, however, that while the credentials are fully auditable, identity statements require inherent trust in something. If the identity infrastructure, e.g., US passports, is not trusted, then we can make no guarantees.

[5] This is, in essence, a specialized zk-rollup [But].

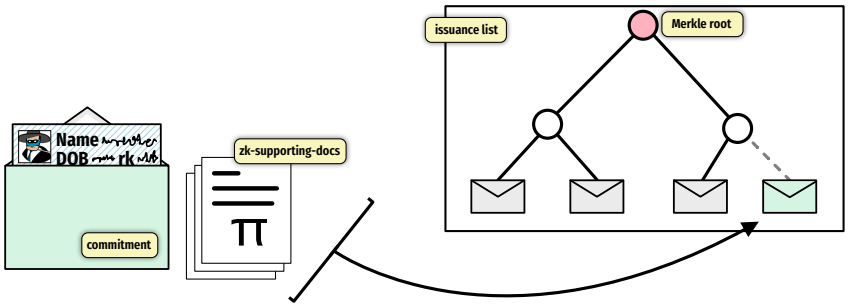

Figure 1: A credential in `zk-creds` is issued by adding it to a Merkle tree after (optionally) presenting zk-supporting-documentation to justify issuance.

tree. We call the set of leaves the *issuance list*. Optionally, protocol designers can require *zero-knowledge supporting-documentation* that the attributes match some (existing) document without revealing any additional information. In our implemented example (see Section 7), this consists of a zero-knowledge proof that the attributes in the credential match a US passport.

We emphasize that, while our example trusts an existing identity document issuer—the passport authority—it requires no additional trusted parties, no existing parties to take on additional trusted roles, or even modification of the issuer. With standard anonymous credentials, we would need to also trust the security of credential signing keys, likely held by an additional party. In our example, we need only some way of maintaining a list of issued credentials.

Clients, once issued a credential, *show* a credential to gain access to some resource, as shown in Figure 2. The client presents a non-interactive zero-knowledge proof that: 1) they have a credential (a commitment) in the Merkle tree of issued credentials and 2) the attributes meet some *access criteria*.

Crucially, the zero-knowledge proof hides which credential is used, the credential's attributes, and the details of how the access criteria were met. In our example, the client uses a credential containing their birth date to show they are over 18 and gain access to a website. The verifier learns only that the client is over 18, not their identity, which credential they used, their exact birth date, or any of the other attributes in their credential.

2.2 Design features

As illustrated by our example application, our approach to building anonymous credentials has a number of important features.

Flexible access criteria. Application developers can define arbitrary access criteria at any point in the lifetime of the system. We support common features from the anonymous credential literature, such as hidden-attribute credentials, inequality or expiry checks, rate limiting, and cloning resistance where violating the rate limit (e.g., by sharing a credential with others) results

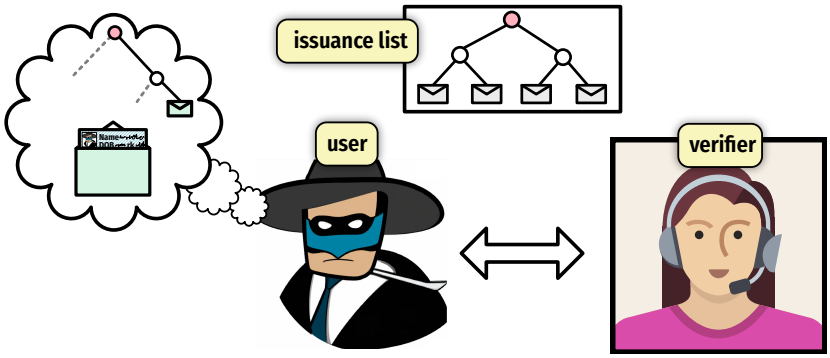

Figure 2: To show a credential in `zk-creds`, the prover uses knowledge of their credential opening and the position of the credential in the issuance list to construct a zero-knowledge proof. The verifier need only know the issuance list root.

in the credential's identification and revocation. And because we support the efficient encoding of access criteria as an NP relation, we can easily support more complex criteria than existing schemes, such as a proof of residency in a given municipality to access ebooks from a local public library.

Auditable issuance. Credential issuance can be publicly auditable as well: e.g., in our passport example anyone can download the list of credentials and, with zk-supporting-documentation, verify both that a credential was issued and why. Even without such documentation, all issued credentials are visible and issuance can be investigated. In contrast, it is impossible to enumerate, let alone audit, every credential signed with a given key.

Flexible credential management. Because credential issuance is simply a matter of list management, credential issuance is flexible. In many cases, we need not find a trusted party at all: a simple bulletin board is sufficient, as is a blockchain. In other cases, a central party can maintain the list without needing to be trusted to secure signing keys.

Signature-issued credentials. Separately, in cases where there is a party who is trusted to issue correct credentials without public auditability and is trusted, willing, and able to secure signing keys, our implementation of `zk-creds` also supports issuing credentials via signatures. In this case, there is no issuance list. When feasible, this leads to faster credential shows and removes the overhead of managing a witness to list membership.

Witness management. Using Merkle trees for credential issuance requires the user to maintain an up-to-date witness to the credential's membership in an issuer's list. Periodically, the user can ask the issuer for an updated witness. Looking ahead, this also lets any user update their witness by downloading logarithmic-sized updates from the tree's *frontier* and a constant number of Merkle roots (see Appendix B).

Revocation. Many existing approaches require expensive asymmetric cryptographic operations for each revocation. Some schemes, like EPID [BL09], require each credential show to perform work linear in the number of revoked private keys. Other schemes use, e.g., RSA accumulators, which require recomputing accumulator witnesses per revocation at cost linear in the number of revocations. And, while more efficient accumulators exist [BBF19], such techniques have not been used for revocation, to the best of our knowledge. In contrast, each revocation in `zk-creds` only requires removing the credential from its Merkle tree, incurring only logarithmic costs in the number of issued credentials. This captures revocation of the credential where the holder's public identity is known, and so called *private key revocation* where a stolen or leaked credential is banned.

3 Preliminaries

3.1 General notation

We write $x := z$ to denote variable assignment, and $y \leftarrow S$ to denote sampling uniformly from a set S. $y := \mathsf{A}(x; r)$ denotes the execution of a probabilistic algorithm A on input x, using randomness r. We write $\overline{x} := x_1, x_2, \ldots$ to denote a variable-length list, and boldface to denote a vector. For an arbitrary, efficiently computable predicate P, we say that *a proof of knowledge of a relation* $R = \{(x; w) : P(x, w)\}$ with respect to an *instance* x is a proof of knowledge of the *witness* w such that $P(x, w)$ is satisfied. We use $\mathsf{Com}(v; r)$ to denote a commitment to the value v with randomness r. The security parameter of our system is denoted by λ.

3.2 Merkle trees

In `zk-creds` we use Merkle trees T to represent set membership. The root of a tree T is denoted T_{root}. A Merkle *forest* F is a set of Merkle tree roots. Merkle trees have the following functionality:

$T.\mathsf{Insert}(v) \to T'$ Inserts the value v into the next free leaf in T and returns the modified tree.

$T.\mathsf{Remove}(v) \to T'$ Removes v from the tree (if present) and returns the modified tree.

$T.\mathsf{AuthPath}(v) \to \theta$ Creates an *authentication path* θ that proves that $v \in T$. The size of θ is proportional to the height of the tree.

3.3 Cryptographic building blocks

We describe two non-interactive zero-knowledge (NIZK) proof systems we use to build `zk-creds`. Both systems operate within a *type-3 non-degenerate bilinear*

group which we denote bg. We define the necessary notation and preliminaries below, and use them in the construction of $\mathsf{LinkG16}$ in Appendix E.

3.3.1 Bilinear groups

We will work exclusively with prime-order groups and their associated scalar fields. Group elements are denoted by capital letters, e.g. $A \in \mathbb{G}$. Following the conventions of both mathematical literature and programming libraries for notational convenience, we use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$, and multiplicative notation for $\mathbb{G}_T$ (with identity denoted by 1).

$\mathsf{bg} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ is a (non-degenerate) *Type-3 bilinear group* if $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are of prime order, there are no efficient group homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$, and $G \in \mathbb{G}_1$ and $H \in \mathbb{G}_2$ are generators. Moreover, the equipped *bilinear pairing* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ must have the following properties:

Bilinearity For all $a, b \in \mathbb{Z}_p$, $X \in \mathbb{G}_1$, and $Y \in \mathbb{G}_2$: $e(aX, bY) = e(X, Y)^{ab}$

Non-degeneracy For any non-trivial $G \in \mathbb{G}_1, H \in \mathbb{G}_2$: $e(G, H) \neq 1$

As a corollary of bilinearity and non-degeneracy, $e(G, H) \in \mathbb{G}_T$ must be a generator. Hereafter, we represent scalar multiplication with some scalar $a \in \mathbb{Z}_p$ by $[a]_1 := aG$ and $[a]_2 := aH$, respectively.

3.3.2 Groth16

We describe a trusted-setup zkSNARK scheme, due to Groth[6] [Gro16], which operates over a non-degenerate type-3 bilinear group. We use $\mathbb{F}$ to denote the scalar field of the bilinear group. At a high level, a Groth16 proof proves that an arithmetic circuit over $\mathbb{F}$ is satisfied by a set of public inputs (values known to the verifier) and private inputs (values not known to the verifier).

$\mathsf{G16.Setup}(\mathsf{bg}, \mathsf{desc}) \to \mathsf{crs}$ Generates a common reference string for the given arithmetic circuit description. crs contains the elements from the bilinear group bg necessary to compute the expressions in $\mathsf{G16.Prove}$ below.

$\mathsf{G16.Prove}(\mathsf{crs}, \{a_i\}_{i=0}^{\ell}, \{a_i\}_{i=\ell+1}^{m}) \to \pi$ Proves the circuit described by crs is satisfied, where $a_0, \ldots, a_\ell \in \mathbb{F}$ represent the circuit's public input wires and $a_{\ell+1}, \ldots, a_m \in \mathbb{F}$ represent the private wires. π is of the form (A, B, C) where $A, C \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

$\mathsf{G16.Verify}(\mathsf{crs}, \pi, \hat{S}) \to \{0, 1\}$ Verifies the proof $\pi = (A, B, C)$ with respect to the *prepared public input* $\hat{S} = \sum_0^\ell a_i W_i$ by checking the relation

$$e(A, B) \stackrel{?}{=} e([\alpha]_1, [\beta]_2) \cdot e(C, [\delta]_2) \cdot e(\hat{S}, H),$$

[6] To support linkage, we diverge slightly from Groth's original construction by setting one of the trapdoor values γ to 1. This does not affect security; zero-knowledge and knowledge soundness were proven by Kohlweiss et al. [KMSV21] for a strictly larger CRS which also sets $\gamma := 1$.

where $[\alpha]_1$, $[\beta]_2$, and $[\delta]_2$ come from crs, and W_i is the crs value whose coefficient represents the value of the i-th wire in the circuit. As shorthand, verification can also be written as $\mathsf{G16.Verify}(\mathsf{crs}, \pi, \mathbf{a})$.

$\mathsf{G16.Rerand}(\mathsf{crs}, \pi) \to \pi'$ Rerandomizes the proof $\pi = (A, B, C)$ by sampling $\zeta, \omega \leftarrow \mathbb{F}$ and computing

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

By Theorem 3 in [BKSV21], the output of Rerand is perfectly indistinguishable from a fresh proof of the same underlying statement.

3.3.3 Groth16 Linkage

We describe a high-level interface that allows us to construct a (blinded) *linkage* proof over Groth16 proofs. This allows one to show that a hidden collection of Groth16 proofs $\pi_1, \ldots, \pi_k$ all share some subset of *hidden common inputs* $\mathbf{x}^*$, not known to the verifier. Concretely, this proof system proves that

$$\bigwedge_{i=1}^{k} \mathsf{G16.Verify}(\mathsf{crs}_i, \pi_i, (\mathbf{x}^*, \mathbf{x}_i))$$

where $\mathbf{x}_i$ are the non-hidden public inputs (i.e., public inputs known to the verifier). See Appendix E for the full description and security proofs of $\mathsf{LinkG16}$. For `zk-creds`, however, it suffices to specify the functionality:

$\mathsf{LinkG16.Link}(\mathbf{x}^*, \{\mathsf{crs}_i, \pi_i\}_{i=1}^k) \to \pi_{\mathsf{link}}$ Constructs a zero-knowledge proof of the above relation, with respect to hidden common inputs $\mathbf{x}^*$.

$\mathsf{LinkG16.LinkVerify}(\pi_{\mathsf{link}}, \{\mathsf{crs}_i, \mathbf{x}_i\}_{i=1}^k) \to \{0, 1\}$ Verifies the above statement with respect to the given public inputs and Groth16 CRSs.

3.4 Cryptographic assumptions

We state the security properties of the above schemes and the cryptographic assumptions necessary to achieve them. For brevity, we defer the definitions of the specific assumptions to the cited references.

Groth16 is perfectly zero-knowledge and weak white-box simulation-extractable against algebraic adversaries under $q-\mathsf{dlog}$ and a linear independence assumption [BKSV21]. We also assume this result holds under the common Groth16 substitution $\gamma = 1$. We use Poseidon [GKK+19] to instantiate a hash for Merkle trees, as well as commitments. Finally, we assume that the key-prefixed Poseidon hash function, used to instantiate the gadgets in Section 5.3, is a PRF. As Poseidon is a sponge construction, prefixing is secure. Separately, see Appendix G for an alternate instantiation using Pedersen hashes and perfectly hiding commitments.

4 Definitions

4.1 Security definitions

Security definitions are given by an ideal functionality in Figure 3, corresponding to the usual security properties of anonymous credentials: unforgeability, correctness and unlinkability. It also implies an additional security property, session binding: shows of a credential are inherently bound to the channel or session in which they are presented, thus preventing replay attacks.

Threat model. The ideal functionality corresponds to the following general threat model. We assume all issuers, verifiers, and (almost all) users are malicious and can collude. We inherit the standard requirement that, for anonymity, there must be at least two honest users with valid issued credentials. We also assume that there exists a reliable mechanism for parties to agree on the list of issued credentials.

4.2 Anonymous credentials

We give a generic overview of the data structures and algorithms which our scheme instantiates.

Let a *credential* be the commitment $\mathsf{cred} := \mathsf{Com}(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs}; r)$ where nk is the *pseudonym key*, a private random value used to generate persistent pseudonyms; rk is the *rate key*, a private random value used to generate rate-limit tokens; $\mathsf{attrs} \in \mathcal{A}$ is an arbitrary set of public and hidden *attributes*; and r is the commitment randomness. Note that the values within the credential remain private by the hiding property of commitment schemes, with the exception of attribute information revealed by the user.

We say that an issuer I *issues* a credential if it appears on I's credential list CL. Looking ahead, while the credential list may be instantiated in many ways (e.g. an accumulator, or Merkle tree), we later instantiate this as a Merkle forest and a list of corresponding Merkle trees $\mathsf{CL} := (F, \overline{T})$, with an *authentication path* θ providing membership attestation. Every issuer has some *issuance criteria* ι that the requester must meet in order to have their cred issued, e.g., that the birth date in cred matches a signed digital passport. Over an issuer-authenticated channel, the requester (running $\mathsf{IssueReq}$) sends cred to the issuer with some *zk-supporting-documentation* sd, e.g., a Groth16 proof or a digital signature, that convinces the issuer of the criteria. The issuer runs $\mathsf{IssueGrant}$ and, upon success, adds cred to their list and returns an authentication path θ attesting to its issuance.

Next, let the list of all possible *access criteria* be $\Phi := \{\phi \mid \phi : \mathcal{A} \to \{0,1\}\}$ which can be defined dynamically by users, verifiers, or even third-parties for an application using `zk-creds`, even after system instantiation. Over an anonymous channel, a user *shows* a credential (running $\mathsf{ShowCred}$) by presenting a zero-knowledge proof that they have a valid issued credential (with θ as

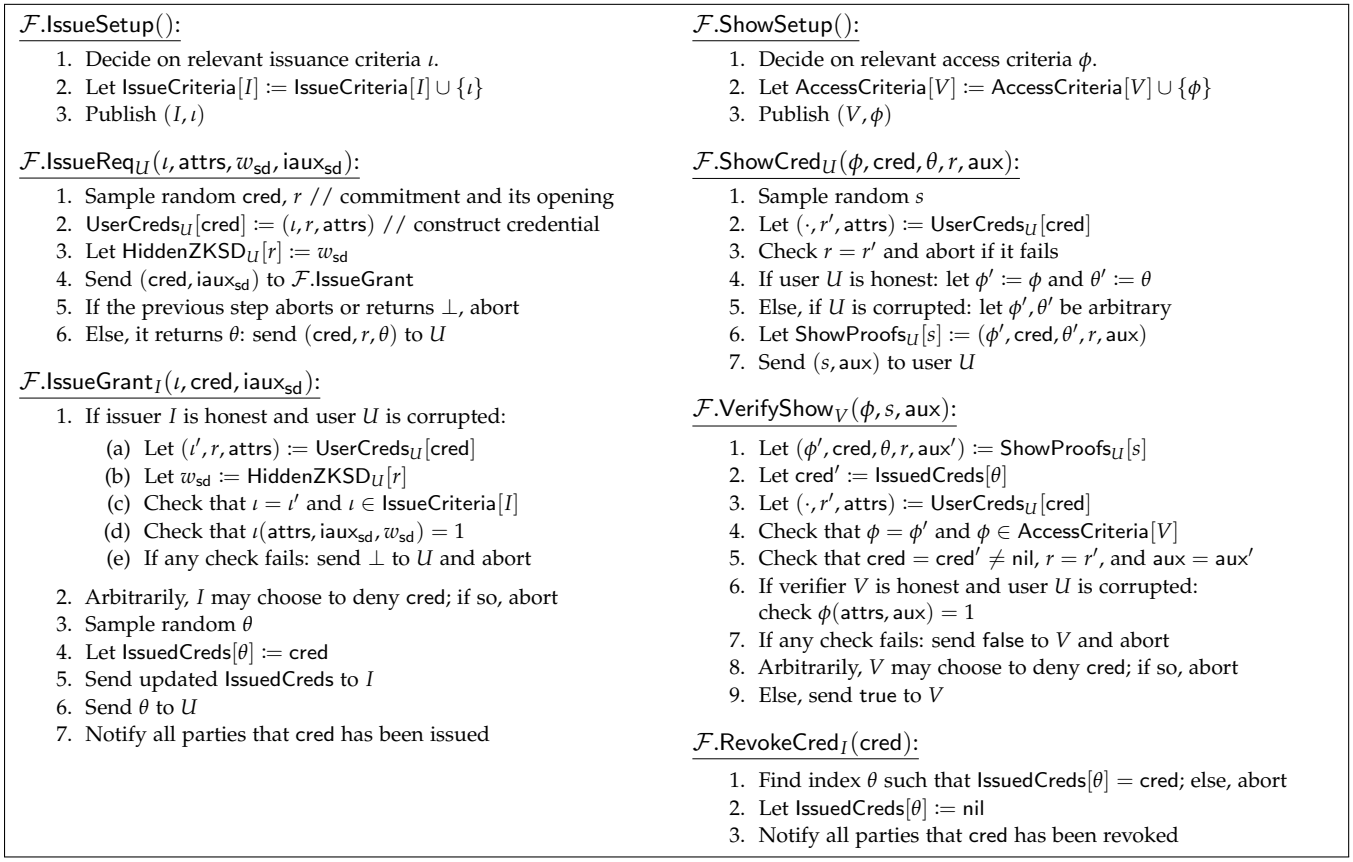

Figure 3: An ideal functionality $\mathcal{F}$ for `zk-creds`.

a private witness) whose attributes in witness w satisfy the verifier's access criteria. The proof must also be bound to some session context, aux. The verifier runs VerifyShow and, upon success, grants access to the user.

An anonymous credential system with zk-supporting-documentation can then be defined by the following algorithms (where the subscript U, I, or V denotes that the user, issuer, or verifier runs the algorithm, respectively):

Setup$(1^\lambda) \to$ pp Generates the system parameters.

IssueSetup(pp) $\to \iota$ Establishes the *public* attribute fields and issuance criterion ι for obtaining a credential.

ShowSetup(pp) $\to \phi$ Establishes a new access criterion $\phi \in \Phi$ for showing a credential.

IssueReq$_U$(pp, ι, attrs, w_{sd}, $\mathsf{iaux}_{\mathsf{sd}}$) $\to$ (cred, sd) Creates and requests a credential cred with zk-supporting-documentation sd under issuance criteria ι.

$\mathsf{IssueGrant}_I(\mathsf{pp}, \iota, \mathsf{CL}, \mathsf{cred}, \mathsf{sd}) \to (\mathsf{CL}', \theta)$ Decides whether to grant a user the requested credential cred; if so, adds it to the list CL and returns its issuance attestation θ.

$\mathsf{ShowCred}_U(\mathsf{pp}, \phi, \mathsf{CL}, \mathsf{cred}, \theta, w, \mathsf{aux}) \to (\pi_{\mathsf{link}}, \mathsf{aux})$ Shows that an issued cred satisfies access criteria ϕ.

$\mathsf{VerifyShow}_V(\mathsf{pp}, \phi, \mathsf{CL}, \pi_{\mathsf{link}}, \mathsf{aux}) \to b$ Validates a credential show, including the current session context in aux.

$\mathsf{RevokeCred}_I(\mathsf{pp}, \mathsf{CL}, \mathsf{cred}) \to \mathsf{CL}'$ Revokes a credential.

5 Construction

`zk-creds` assumes there is a list of issued credentials maintained by either a trusted party, some Byzantine system, or a blockchain. Our scheme provides three sets of functionalities: issue, show, and revoke. Through the issuance process, a user, Alice, convinces the issuance mechanism she should be given a credential. Once her credential is put on the issuance list, Alice can then use the credential to show she meets some access criteria. Conceptually, using a credential involves two steps: (1) a membership proof that the credential is on the issuance list, and (2) a proof that the committed attributes meet some access criteria. Finally, an issuer is able to revoke a credential if need be by simply removing it from the list.

We can realize this paradigm in different ways and using different set-membership techniques such as an RSA accumulator, purpose built zero-knowledge schemes [ZBK+22], or even using signatures of issuance.

In our construction of `zk-creds`, we realize membership proofs using Merkle forests, a new approach that allows developers to trade a slight increase in verification time and witness data for a large reduction in proving time. For access criteria checks, we provide developers with a set of *gadgets*. Gadgets can be composed to form complex access criteria checks. Finally, we tie these components together with a new blind Groth16 proof, of potentially independent interest, that lets us show multiple Groth16 proofs shared the same blinded input without—as in commit-and-prove—creating a persistent identifier. This allows us to reuse the membership proof across multiple credential shows without the reused proofs being tracked.

We now give details on our specific instantiation and describe the full construction in Figure 4.

5.1 Merkle forests

Rather than using a single Merkle tree to accumulate credentials and prove membership, `zk-creds` uses a forest of Merkle trees. The membership proof attests to two parts: $\mathsf{cred} \in T$ for some Merkle tree T, and $T_{\mathsf{root}} \in F$ where F is the forest of Merkle trees containing issued credentials. Compared to a single

$\underline{\mathsf{Setup}}(1^\lambda, h, n)$:

1. Choose bilinear group bg w/ large prime order p
2. Let desc_T be a circuit w/ public inputs $(T_{\mathsf{root}}, \mathsf{cred})$, where T is a Merkle tree of height h. It asserts that there is an auth path θ attesting to $\mathsf{cred} \in T$
3. Let desc_F be a circuit w/ public inputs $(T_{\mathsf{root}}, \mathsf{cred}, F)$, where F is a forest of n Merkle trees. It asserts that the tree is in the forest: $(T_{\mathsf{root}} = F_1) \vee \ldots \vee (T_{\mathsf{root}} = F_n)$
4. Compute $\mathsf{crs}_T := \mathsf{G16.Setup}(\mathsf{bg}, \mathsf{desc}_T)$
5. Compute $\mathsf{crs}_F := \mathsf{G16.Setup}(\mathsf{bg}, \mathsf{desc}_F)$
6. Let $\mathsf{pp} := (\mathsf{bg}, \mathsf{crs}_T, \mathsf{crs}_F)$
7. Return pp

$\underline{\mathsf{IssueSetup}}(\mathsf{pp})$:

1. Decide on issuance criteria $(\iota_{\mathsf{zk}}, \iota_{\mathsf{pub}})$ for supporting docs
2. Let desc be a circuit w/ public input $\mathsf{iaux}_{\mathsf{zk}}$ asserting: $\iota_{\mathsf{zk}}(\mathsf{attrs}, \mathsf{iaux}_{\mathsf{zk}}) = 1 \wedge \mathsf{cred}$ opens to $(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs})$
3. Compute $\mathsf{crs}_\iota := \mathsf{G16.Setup}(\mathsf{bg}, \mathsf{desc})$
4. Return $(\mathsf{crs}_\iota, \iota_{\mathsf{pub}})$

$\underline{\mathsf{ShowSetup}}(\mathsf{pp})$:

1. Decide on access criteria ϕ for satisfying attributes
2. Let desc be a circuit w/ public inputs $(T_{\mathsf{root}}, \mathsf{cred}, \mathsf{aux})$ which asserts: $\phi(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs}, \mathsf{aux}) = 1 \wedge \mathsf{cred}$ opens to $(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs})$
3. Compute $\mathsf{crs}_\phi := \mathsf{G16.Setup}(\mathsf{bg}, \mathsf{desc})$
4. Return crs_ϕ

$\underline{\mathsf{IssueReq}}(\mathsf{pp}, \mathsf{crs}_\iota, \mathsf{attrs}, w_{\mathsf{sd}}, (\mathsf{iaux}_{\mathsf{zk}}, \mathsf{iaux}_{\mathsf{pub}}))$:

1. Sample r, nk, rk // commitment nonce, pseudonym key, rate key
2. Commit $\mathsf{cred} := \mathsf{Com}(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs}; r)$
3. Let $w := (w_{\mathsf{sd}}, r, \mathsf{nk}, \mathsf{rk}, \mathsf{attrs})$ // collect the witnesses
4. Prove $\pi_\iota := \mathsf{G16.Prove}(\mathsf{crs}_\iota, (\mathsf{cred}, \mathsf{iaux}_{\mathsf{zk}}), w)$
5. Let $\mathsf{sd}_{\mathsf{zk}} := (\pi_\iota, \mathsf{iaux}_{\mathsf{zk}})$ // collect the supporting docs
6. Let $\mathsf{sd}_{\mathsf{pub}} := \mathsf{iaux}_{\mathsf{pub}}$
7. Send $(\mathsf{cred}, \mathsf{sd}_{\mathsf{zk}}, \mathsf{sd}_{\mathsf{pub}})$ to issuer
8. Receive the root of the modified credential tree T' and a Merkle auth path θ attesting to $\mathsf{cred} \in T'$
9. Store $(\mathsf{nk}, \mathsf{rk}, r, \mathsf{attrs}, \theta)$

$\underline{\mathsf{IssueGrant}}(\mathsf{pp}, (\mathsf{crs}_\iota, \iota_{\mathsf{pub}}), (F, \overline{T}), \mathsf{cred}, (\mathsf{sd}_{\mathsf{zk}}, \mathsf{sd}_{\mathsf{pub}}))$:

1. Parse $(\pi_\iota, \mathsf{iaux}_{\mathsf{zk}}) := \mathsf{sd}_{\mathsf{zk}}$
2. Check $\iota_{\mathsf{pub}}(\mathsf{sd}_{\mathsf{pub}})$ // check public supporting docs
3. Check $\mathsf{G16.Verify}(\mathsf{crs}_\iota, \pi_\iota, (\mathsf{cred}, \mathsf{iaux}_{\mathsf{zk}}))$ // check ZK SD
4. If either check fails, reject issuance and abort
5. Else, choose T from forest F and let $T' := T.\mathsf{Insert}(\mathsf{cred})$
6. Let $\theta := T'.\mathsf{AuthPath}(\mathsf{cred})$ // θ attests to $\mathsf{cred} \in T'$
7. Store T' and update the forest F
8. Send θ to user U

$\underline{\mathsf{ShowCred}}(\mathsf{pp}, \overline{\mathsf{crs}}, (F, T_{\mathsf{root}}), \mathsf{cred}, \theta, \{w_i, \mathsf{aux}_i\}_{i=1}^k)$:

1. Parse $(\{\mathsf{crs}_{\phi_i}\}_{i=1}^k, \mathsf{crs}_T, \mathsf{crs}_F) := \overline{\mathsf{crs}}$ and cred's $T_{\mathsf{root}} \in F$
2. For all $i = 1, \ldots, k$, compute the access criteria proof: $\pi_{\phi_i} := \mathsf{G16.Prove}(\mathsf{crs}_{\phi_i}, (T_{\mathsf{root}}, \mathsf{cred}, \mathsf{aux}_i), w_i)$
3. Prove $\pi_T := \mathsf{G16.Prove}(\mathsf{crs}_T, (T_{\mathsf{root}}, \mathsf{cred}), \theta)$ // tree
4. Prove $\pi_F := \mathsf{G16.Prove}(\mathsf{crs}_F, (T_{\mathsf{root}}, \mathsf{cred}, F), \mathsf{nil})$ // forest
5. Let $\overline{\pi} := (\{\pi_{\phi_i}\}_{i=1}^k, \pi_T, \pi_F)$ // collect the proofs
6. Prove $\pi_{\mathsf{link}} := \mathsf{LinkG16.Link}((T_{\mathsf{root}}, \mathsf{cred}), \{\mathsf{crs}_i, \pi_i\}_{i=1}^{k+2})$
7. Send $(\pi_{\mathsf{link}}, \{\mathsf{aux}_i\}_{i=1}^k)$ to verifier V

$\underline{\mathsf{VerifyShow}}(\mathsf{pp}, \overline{\mathsf{crs}}, F, \pi_{\mathsf{link}}, \{\mathsf{aux}_i\}_{i=1}^k)$:

1. Parse $(\{\mathsf{crs}_{\phi_i}\}_{i=1}^k, \mathsf{crs}_T, \mathsf{crs}_F) := \overline{\mathsf{crs}}$
2. Let $(\mathsf{aux}_{k+1}, \mathsf{aux}_{k+2}) := (\mathsf{nil}, F)$ // collect auxiliary inputs
3. Check $\mathsf{LinkG16.LinkVerify}(\pi_{\mathsf{link}}, \{\mathsf{crs}_i, \mathsf{aux}_i\}_{i=1}^{k+2})$
4. Upon success, accept. Else, reject

$\underline{\mathsf{RevokeCred}}(\mathsf{pp}, (F, \overline{T}), \mathsf{cred})$:

1. Find $\mathsf{cred} \in T$ within forest F; if cred not found, abort
2. Let $T' := T.\mathsf{Remove}(\mathsf{cred})$
3. Store T' and update the forest F
4. Return T'

Figure 4: `zk-cred` Construction. NB: Although the inputs $(T_{\mathsf{root}}, \mathsf{cred})$ are public in all Groth16 proofs in ShowCred, they are hidden from the verifier by LinkG16. Also note that any necessary updates to the auth path θ or credential list $(F, \overline{T})$ are handled out-of-band.

Merkle tree, the Merkle forest approach gives us a tunable tradeoff between proving time and verification time. Shorter Merkle trees can drastically reduce proving costs—up to 50%, or 143ms (see Appendix B). Furthermore, since forest membership is a simple OR-proof over Merkle tree roots, the cost of a larger forest is negligible to the prover and allows for a much larger list. Also, we note that, for the size of the forests we consider, the additional verification cost is trivial (137μs).

Witness management. Showing a credential in `zk-creds` requires knowing the witness (a.k.a., authentication path) θ attesting to issuance in its Merkle tree. θ must be updated as credentials are added or removed from the tree. In a naïve construction, a user might download newly-issued credentials to update the tree. However, this requires the user to construct and maintain a local copy of their entire Merkle tree, which is often impractical. On the other extreme, users could periodically query an issuer or list manager to provide the updated θ. However, this uniquely identifies the credential and strongly correlates with subsequent shows, posing a large privacy risk (especially in low-use deployments). We introduce better constructions leveraging Merkle forests in Appendix B.

5.2 Blind Groth16

The membership proof is the most costly part of ShowCred. Looking ahead, it takes 460ms to complete. While the access criteria check must be redone for every show in many cases—for example, rate-limited shows include a token that uniquely identifies reuse—the membership proof does not change unless more credentials are issued.

We use a blind Groth16 linkage proof to combine a membership proof with (perhaps multiple) access proofs. Blind Groth16 lets us reuse an already computed membership proof in multiple shows without breaking privacy. Furthermore, it expands the functionality of the system by supporting the easy composition of access criteria: without this ability to compose access criteria, system designers would need to either: 1) dynamically generate circuit parameters for gadgets as they are needed, 2) determine in advance all the gadgets they will support, or 3) generate the circuit parameters for every combination of gadgets that could be used.

Concretely, blind Groth16 lets us prove that a number of Groth16 proofs are all made with respect to the same credential *without* revealing the credential. At a high level, the algorithm works as follows. First, it prepares the public inputs (here, cred and its Merkle root) shared by the underlying proofs. For each proof, it then blinds a copy of the prepared input, and blinds the proof in a way that cancels with the blinded input. Finally, it proves that all the blinded inputs are consistent with each other. After canceling the blinding factors, the verification equation is identical to the typical Groth16 verification equation.

For more detail, see the description of LinkG16 in Appendix E.

5.3 Gadgets

Since ShowCred supports arbitrary statements, verifiers have the flexibility to add and remove helpful subcircuits, or *gadgets*, from their protocol. In fact, rather than embedding gadgets in existing circuits, verifiers can make use of the structure of ShowCred to create a separate proof for each gadget and link them together. The benefit to this kind of customization is twofold: users can precompute and cache standalone gadget proofs separately from other access criteria proofs, and verifiers are freed from having to define custom circuits and generate the CRSs.

Gadgets are arbitrary NP relations which can capture nearly any conceivable identity check. We now describe some gadgets that serve as building blocks for `zk-creds`-based systems. Recall that rk denotes the rate key, used for generating rate-limit tokens, and nk denotes the pseudonym key, used for deriving uniform but linkable tokens.

Linkable show Reveals a pseudonym $\mathsf{PRF}_{\mathsf{nk}}(\mathsf{ctx})$ that persists across interactions in a given context ctx, but is unlinkable to any use of the credential in other contexts. For example, a single Sybil-resistant credential could be used for creating unlinkable accounts across sub-forums within a single site, such as Discord servers or subreddits.

Rate limiting Limits users to performing ShowCred only N times per epoch, for some verifier-chosen rate limit N. Every ShowCred, the user produces a pseudorandom token $\mathsf{tok} = \mathsf{PRF}_{\mathsf{rk}}(\mathsf{epoch} \| \mathsf{ctr})$, reveals epoch, and proves that ctr is less than N.

Cloning resistance Performs the same function as rate limiting, but deanonymizes rate violators. The technique was introduced by Camenisch et al. [CHK+06] (Section 5.2). Every run of ShowCred, the user receives a nonce from the verifier and sends two tokens:

$$\mathsf{tok}_1 = \mathsf{PRF}_{\mathsf{rk}}(\mathsf{epoch} \| \mathsf{ctr})$$
$$\mathsf{tok}_2 = \mathsf{id} + H(\mathsf{nonce}) \cdot \mathsf{PRF}'_{\mathsf{rk}}(\mathsf{epoch} \| \mathsf{ctr})$$

where id is an identifying attribute (e.g., credential hash). As above, ShowCred proves the tokens are constructed correctly. If one of these shows is reused, tok_1 will be repeated, but tok_2 will be distinct, giving the verifier two instances of the tok_2 equation and two unknown variables: id and $\mathsf{PRF}'_{\mathsf{rk}}(\mathsf{epoch} \| \mathsf{ctr})$. Solving the equation for id identifies the credential holder. Note that if id is the credential (or its hash), then the cloned credential can be revoked immediately by removing it from the issuance list.

Expiry Opens an attribute e in the credential and proves that $e > \mathsf{today}$, i.e., that the credential is not yet expired.

Session binding Gives the verifier the ability to reject replayed ShowCred proofs by binding a verifier-chosen nonce, or *session context*, to every ShowCred. This can be done by including an empty proof that takes the nonce as the public input[7].

Join Allows credentials to be composed by *joining* them along some common attribute(s), such as full name or address. This is either done inside a single ShowCred, or between separate ShowCreds by publicly committing to the common attribute(s).

5.4 Signature-issued credentials

One of the major advantages of `zk-creds` is that we do not need issuers who are trusted to hold signing keys and can instead use a transparency log or blockchain to issue credentials. However, if this feature is not needed, we can build a traditionally issued anonymous credential scheme that retains `zk-creds`' other features and flexibility. Because our gadget-based approach is flexible, we can replace the membership-check gadget with one that instead verifies a signature. This means `zk-creds`, like Coconut [SAB+19], also supports credentials that are issued by signing under a standard signature (e.g., ECDSA or Schnorr) either by a single party or by a threshold of parties via a threshold signature scheme such as FROST [KG21]. Moreover, we can compose credentials issued via a list with ones issued via signatures.

5.5 Additional features

Notably, our construction of `zk-creds` also allows for the construction of protocols with several features previously only available in dedicated schemes.

Hidden issuer We can completely hide the identity of a credential issuer, e.g., in situations where leaking where a credential came from can cause significant harm to privacy. While this is not a new notion in the literature, very few existing schemes support this hidden issuer property. `zk-creds` supports this inherently, as ShowCred can be performed with respect to synthetic lists created by concatenating lists maintained by different issuers, thus hiding the issuer. One drawback though is that multiple issuers will likely issue different credential formats.

Hidden credential type `zk-creds` can also be configured to hide the credential type which is both independently useful as well as necessary to fully support

[7] This binds the nonce to the Groth16 proof, assuming some basic properties about the circuit [BKSV21].

hidden issuers. In `zk-creds`, credentials with different attributes can be padded to the same size post-issuance, and then used interchangeably for and efficiently verified over an access criterion. This is accomplished by constructing a new circuit that is the OR of the criteria on the individual credentials.

Delegation Issuance authority is delegatable in `zk-creds`. Authority to issue a credential can be shown via zk-supporting-documentation that is itself the show of another credential. Moreover, because the proof in zk-supporting-documentation is general-purpose, the delegation process can constrain attributes in the credential being issued. For example, we could define a credential for an authority that can only be delegated three layers deep by having a hidden attribute of delegation level decremented each time. Credential attributes can be selectively delegated as well.

5.6 NIZK setup

Groth16 requires a one-time trusted setup to generate a set of parameters called a common reference string (or CRS) for each statement (a.k.a., circuit). Once this CRS is generated, it can be used throughout the lifetime of the system to prove different instances of the statement.

Distributed setup for Groth16 CRSs is a solved problem via multiparty computation setup *ceremonies* [BGM17, BCG+15, KMSV21] that need only two honest parties. These have been run with hundreds of users and used to secure billions of dollars in cryptocurrency. These protocols are efficient and produce *subversion-resistant zero-knowledge proof systems* [Fuc18]—systems which ensure that, even if all parties in a setup are malicious, the proofs are still zero-knowledge and user privacy is unaffected.

Independently, as mentioned previously, we have also sought to minimize the impact of this CRS on the flexibility of `zk-creds`. By utilizing blind Groth16, a system designer does not need to decide on and pre-generate CRSs for all possible combinations of access criteria they wish to support and can instead just generate CRSs on a per-gadget basis.

5.7 Security argument

We argue that `zk-creds` is secure under computationally-bounded adversaries performing *static* corruption of users, verifiers, and even issuers who can collude arbitrarily with other parties. Without loss of generality we consider the security of our protocol under a single issuer, since any corrupted issuer can potentially corrupt the credential list; see Section 8 for a discussion of various methods to mitigate and distribute trust in a multi-issuer setting. Let the ideal functionality $\mathcal{F}$ represent the algorithms defined in Figure 3.

The simulator $\mathcal{S}$ first runs Setup and gets the necessary trapdoors for the simulation and extraction of zero-knowledge proofs in Setup, IssueSetup, and ShowSetup. $\mathcal{S}$ maintains a table mapping credentials in its simulated real-world protocol to their ideal counterparts in $\mathcal{F}$ and, when necessary, updates the mapping when it needs to create a real-world object for an ideal-world one (or vice versa).

Because all messages between parties in our security game are accompanied by zero-knowledge proofs, the simulator can extract on all adversarially generated messages (e.g., for IssueReq or ShowCred), look up the corresponding ideal-world credentials in its table, and proxy the requests to the ideal functionality. Similarly, for any honest interactions in the ideal functionality, the simulator can model the adversary's view of the real-world protocol by simulating the zero-knowledge proofs with respect to random commitments and, in the case of rate limiting, PRF outputs.

We note that care must be taken in two cases. First, both honest and corrupted issuers can deny credential issuance, so the simulator's table needs to be updated only on successful issuance. Second, both honest and corrupted parties can revoke credentials, so the simulator must also synchronize revocations.

The security of `zk-creds` rests on the assumption that Groth16 is perfectly zero-knowledge and simulation-extractable (in the Algebraic Group Model); that linkage proofs are sound and zero-knowledge under the Random Oracle Model and dlog (see Appendix E); that the Poseidon hash function is collision-resistant and a secure PRF and; for other variants of our scheme, that Schnorr signatures are unforgeable under dlog, Pedersen hashes are collision-resistant under dlog, and Pedersen commitments are computationally binding under the same assumption and perfectly hiding. Then, the adversary's view of the real-world protocol is indistinguishable from a simulated view where honest parties use the ideal functionality $\mathcal{F}$. Therefore, the instantiation of `zk-creds` given in Figure 4 is secure against malicious adversaries who engage in static corruption.

6 Implementation and evaluation

We now detail the evaluation of `zk-creds`.

6.1 Code and setup

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

Client-opt. (C)	ShowCred			VerifyShow			Proof Size	
Server-opt. (S)	C	C (full)	S	C	S	S (batch)	C	S
Simple Possession	5ms	465ms	450ms	3ms			744B	
Expiry	53ms	526ms	461ms					
Linkable Show	43ms	494ms	457ms	5ms	1.5ms	1.8 verifs/ms	1064B	192B
Rate Limiting	60ms	507ms	461ms					
Clone Resistance	90ms	542ms	530ms					

Table 1: Gadget microbenchmarks using Poseidon hashes for two versions of `zk-creds`. Membership proofs are done on a Merkle forest of size 2^{31} (tree height = 24, #trees = 2^8). The first configuration (C) minimizes client-side proving cost; the second configuration (S) maximizes server throughput. ShowCred (full) gives the cost of including membership recomputation while showing a credential. VerifyShow (batch) gives throughput for verifying a set of 100 proofs. We emphasize that all verification numbers are *single-threaded*, allowing for efficient concurrent processing.

Client-opt.	ShowCred	VerifyShow	Proof Size
Simple Possession	2ms	2ms	424B
Expiry	50ms		
Linkable Show	41ms	5ms	744B
Rate Limiting	58ms		
Clone Resistance	88ms		

Table 2: Benchmarks for `zk-creds` configured for minimizing client side proving cost, using signature-based issuance. Unlike the client-optimized benchmarks in Table 1, which uses list-based membership, there is no distinction between a partial or full ShowCred, since the membership proof (a signature verification proof) never has to be recomputed.

Code. `zk-creds` consists of 7.6k lines of Rust code[8] and relies on the Arkworks [Ar22] zkSNARK framework. For benchmarks and statistics, we used the Criterion-rs crate. In addition, we modified an existing Android passport scanner app to extract intermediate cryptographic values from the passport and dump them to a JSON file.[9] The Rust code uses the dump file format for all its passport proofs.

Statistics. Each figure and plot shows the median runtime of 100 executions. Over all experiments, the maximum estimated relative standard error of the median is 1.8%. For completeness, our plots include error bars indicating the 95% confidence interval, though they are not visible due to the low error.

Instantiating cryptographic primitives. We set $\lambda = 128$. We compute Groth16 proofs over the BLS12-381 curve [Bow17b]. The collision-resistant hash function used in all Merkle trees are domain-separated instances of the Poseidon hash function [GKK+19], configured to be compatible with BLS12-381 and a 128-bit

[8]Code repository: `https://github.com/rozbb/zkcreds-rs`
[9]Code repository: `https://github.com/rozbb/zkcreds-passport-dumper`

security level ($\alpha = 3$ and capacity $= 1$). We instantiate Com using key-prefixed Poseidon as well. Separately, in Appendix G, we give benchmarks for an instantiation with provably secure Pedersen hashes and commitments.

6.2 Microbenchmarks

In this section we measure performance of various common gadgets, with reasonable parameter choices. Recall the performance of `zk-creds` depends on credential list size, attribute size, criteria complexity, and number of standalone gadget proofs. We measure the effects of these parameters in greater detail in Appendix A. We also measure the sizes of the associated proving and verifying keys in Appendix H.

Table 1 gives a summary of `zk-creds`'s performance for common usage scenarios. We assume a setting where 2^{31} credentials have been issued. The first variant (C) minimizes client-side proving cost by separating the Merkle forest proof, allowing for reuse across shows. A basic show takes 5ms to produce and 3ms to verify, assuming precomputation of the Merkle tree and forest membership proofs. More complex statements like rate-limited credentials with clone resistance take 90ms to show and 5ms to verify. If the Merkle tree membership proof is not precomputed, then the full show takes an additional 460ms but verification time is unchanged.

The Simple Possession benchmark shows the prover has an attribute-less credential on a list and proves no predicates. This maps to a use case such as possessing a valid access card, as that is often sufficient to enter a building. The remaining benchmarks build on Simple Possession, adding their own predicate to the set of linked proofs. For example, Expiry proves possession, but also bears a single attribute and proves that it has a value less than some timestamp.

Separately, we give an alternate variant of `zk-creds` in Table 1 (S) which is optimized for verification latency and server throughput. The server-optimized construction combines the Merkle membership and criteria check circuits into a single monolithic zkSNARK without proof reuse. As a result, clients pay approximately the full ShowCred cost every time, but since proofs are a single Groth16 proof rather than a linkage proof, they can be batch-verified by the server at 1.8 verifications per millisecond per core. We note it may be possible to batch verify the non-optimized scheme as well, but throughput would be lower as there are at least three times as many proofs.

Finally, to demonstrate the full flexibility of our approach, we also provide a signature-based variant of `zk-creds` in Table 2, where credentials are issued in a more traditional signature-based manner by a trusted issuer or threshold quorum of issuers. We implemented a signature gadget for checking Schnorr (and therefore FROST threshold [KG21]) signatures and measured the proving time to be 129ms (compare to 460ms for tree-based issuance).

Memory usage. Peak memory usage for any of our experiments was 824

MB. Since prover memory usage is a function of circuit size, this peak was, as expected, for the server optimized configuration with the larger monolithic circuit. Separately, the client optimal approach has a peak usage of 800 MB, which arose when proving LinkG16.

7 Case studies: `zk-creds` as a toolkit

We design, implement, and benchmark two full scenarios for `zk-creds` using credentials derived from existing government identity infrastructure without any modifications or coordination. Many government identity documents now include the ability to perform various authentication protocols (e.g, the German and Estonian [Fed, e-E] smart-card enabled national IDs). For our applications, we use US passports, which contain a signed digital copy of the passport's basic data in an NFC-readable chip. Our applications validate that `zk-creds` can be used as a toolkit by application developers to support privacy-preserving identity in realistic applications with complex, compound access criteria.

7.1 Digital passport data

Over 150 countries issue passports with an NFC-readable chip which is standardized in ICAO Doc. 9303 [Rea, ICA]. We are interested in the first two *data groups* on the chip. Data group 1 (DG_1) contains the textual info available on the passport's data page: name, issuing state, date of birth, and passport expiry. Data group 2 (DG_2) contains a JPEG-encoded image of the passport holder's face. The ICAO standard also requires the immutable part of the chip's contents to be signed by the issuing state. For example, every US passport has an RSA PKCS#1 v1.5 signature under a known US State Department public key.

zk-supporting-documentation for passports. While we could just reveal the signed passport to the credential issuer, this 1) requires the issuer be trusted to maintain the confidentiality of sensitive information, and 2) is wholly incompatible with issuance via a bulletin board or blockchain. Instead, we design and implement a zero-knowledge proof that the attributes of a credential commitment match the signed contents of DG_1 and DG_2.

Parsing the contents of an e-passport in zero-knowledge is non-trivial: the signature is not just over DG_1 and DG_2, but the *econtent hash*, which is calculated over the mostly variable-length data groups $\mathsf{DG}_1, \ldots, \mathsf{DG}_{16}$. Variable length inputs are particularly challenging to parse with a fixed-size zero-knowledge circuit. Our zero-knowledge proof is made possible by realizing that the econtent hash is actually a tree hash, roughly of the form

$$H(H(H(\mathsf{DG}_1) \| H(\mathsf{DG}_2) \| \ldots \| H(\mathsf{DG}_{16})) \ldots).$$

We do not care about the contents of DG_3 through DG_{16}, so their hashes can be used directly as witnesses to the proof. $H(\mathsf{DG}_2)$ is the image hash, which can either be hidden as a witness or revealed by showing all of DG_2.[10] Since DG_1 contains the attributes we care about, we must parse DG_1 inside the zero-knowledge proof. Luckily, since DG_1 mirrors the content of the passport's Machine Readable Zone (MRZ), it is fixed-length. The proof then hashes the data group digests along with other fixed-length values until it has computed the econtent hash. We avoid the cost of checking the RSA signature by simply revealing it and the econtent hash.[11] Computing this proof takes less than 2 seconds.

Why zk-proofs over passports are insufficient. A folklore solution for anonymous credentials is to replace the signature with a zero-knowledge proof of signature possession. Consider, for example, Cinderella, [DFKP16]. Cinderella elegantly converts existing X.509 certificates into anonymous credentials via an intricate zero-knowledge proof that validates the X.509 certificate and its entire certificate chain. Assuming identity documents with X.509 certificates, this is seemingly a viable approach for proof of age.

Zero-knowledge proof of possession of an existing signed document is both insufficient in our case and, more subtly, unnecessarily complex. A proof over a certificate or indeed any existing credential is insufficient: it may not include the necessary data and that data cannot be hidden from the issuer. This is a challenge for, e.g., the random seed we use for cloning resistance. As a result, we must issue our own standalone credentials and can, at best, use existing certificates as supporting documentation to justify issuance. But a zero-knowledge proof for supporting documentation has a subtly different goal from the folklore solution; while we must hide the attributes in the certificate and issued credential (e.g., name and date of birth), we can leak other information such as who issued the credential (e.g., the US State Department). While this could uniquely identify a user during a credential show, revealing the same information during issuance poses no such risk: issuance is by definition unlinkable to show anyway. This drastically simplifies the zero-knowledge proof and removes unnecessary complexity.

7.2 Instantiating an issuance bulletin board

An instantiation of `zk-creds` requires a publicly accessible bulletin board to distribute the credential list, as well as parties running our software. We stress

[10] In reality, DG_2 contains slightly more than the bare image; it also has the image's creation and expiry dates.

[11] Revealing the econtent hash reveals the identity of the requester to the passport authority during issuance. However, since ShowCred proofs are unlinkable to issuance, this is not a problem. Moreover, it is possible to hide the hash and verify the RSA signature in the zkSNARK, at a cost of about 868k constraints using the approach from [KZM+16], or 3.6s in our benchmarking environment.

that `zk-creds` can be deployed either on a blockchain or (at much lower cost) a more centralized transparency log; we choose to instantiate the bulletin board here as a smart contract on an Ethereum Virtual Machine (EVM)-compatible blockchain to demonstrate that full decentralization is feasible and because, unlike many other consensus systems in development or deployment, the EVM has comparatively robust development tooling and documentation.

The smart contract stores a list of credentials and corresponding zk-supporting-documentation—together, referred to as an *issuance request*—as well as the current roots of the Merkle forest. An issuer issues a credential by posting the full issuance request to the smart contract. This allows any external auditor to download the full list, reconstruct a local copy of the Merkle forest, then verify in zero-knowledge that each credential was validly issued. While any party (including the user) can audit the full list themselves if desired, they need not do so if they trust another party (e.g., an issuer or auditor) to promptly update and verify the credential list for them. Periodically, users request their credential's authentication path and the updated root from the bulletin board or auditing party. To perform VerifyShow, a verifier only needs to retrieve the current Merkle tree roots F from the bulletin board.

Furthermore, we must prevent DoS attacks from blocking or flooding additions to the bulletin board. Our prototype instantiation assumes a smart contract operator who is authorized to add to the bulletin board contract on behalf of the issuer. An alternative solution, allowing for operator-free setup, is to verify all zk-supporting-documentation and Merkle tree root computations within the EVM smart contract itself. While this is feasible [Torb], verifying proofs and Merkle tree updates on-chain without extensive optimizations is expensive. Another solution is to instead support on-chain proof verification with an optimistic rollup [Eth]: bulletin board additions include a deposit which is burned if, after the smart contract evaluates a challenge, it determines that the proofs or computed Merkle tree roots are invalid. While the challenge still requires costly computation, this is not paid for during issuance.

We implement our smart contract in Solidity and the requisite client-side scripts to post and retrieve data in `web3.js`. Note that we can deploy our contract to any EVM-compatible chain, such as Avalanche's C-Chain or Ethereum itself. EVM contract operations are measured in *gas*, which roughly acts as a complexity-weighted count of the EVM instructions used. Posting each issuance request costs $576,808$ gas, while posting a Merkle root costs $78,355$ gas. The price of gas and the underlying currency it is priced in is highly volatile. As of early June 2022, Ethereum gas prices range from about 20 to 50 Gwei (a Giga-wei is 10^{-9} ETH), and 1 ETH is about \$1800 USD; as such, posting an issuance request costs about \$20–\$50 USD. Gas prices are proportionally similar for Avalanche but, at \$25 USD per token, actual costs would be 70 times smaller.

7.3 Credentials from existing identity infrastructure

Given a construction of zk-supporting-documentation for a passport and a choice of bulletin board in an Ethereum smart contract, application developers can now readily build access control schemes. Once the credentials are issued, multiple developers can independently rely on them by either composing existing identity gadgets or defining new ones.

Issuance. We provide a toolchain to convert a passport into an anonymous credential. An Android app extracts the NFC passport data and a separate program converts it into a credential containing the holder's nationality, full name, and date of birth (dob); a rate key (rk); the passport expiry date (expiry); and the hash of the image of the holder's face (facehash). Separately the program computes the zk-supporting-documentation that this credential is correct with respect to the signature and econtent hash.

The IssueReq payload sent to the issuer consists of the signed econtent hash, credential, and zk-supporting-documentation proof. IssueGrant verifies the proof and econtent hash signature. Upon success, the issuer adds the credential to their list and returns a Merkle authentication path.

Scenario 1: Viewing age-restricted content on the internet. Age-restricted content is common on the internet. For example, in Switzerland, the EU, and the UK, YouTube requires users to upload an image of their ID or credit card in order to prove their age [Gooa]. In this scenario we demonstrate the feasibility of `zk-creds` for proving age without revealing any personally identifying information.

Our `zk-creds` toolkit has three features that are crucial to building a real-world feasible age verification credential. First, it can be used without coordination with existing identity infrastructure. Second, it can readily support other identity credentials, provided they indicate date of birth and are signed. Third and, most crucially, it can create credentials that are clone-resistant (via the gadget in Section 5.3) with easy revocation of cloned credentials. This last point is essential: while a zero-knowledge proof over a passport is itself an anonymous credential,[12] practical usage demands cloning protections. And cloning resistance requires a rate key to be bound to the credential and kept secret from the issuer. Existing identity documents (such as a passport) lack such a key. `zk-creds` allows composition of identity without coordinating with the passport issuer or any trusted party to add such information.

Given issued credentials via passports, building a privacy-preserving age verification scheme with `zk-creds` is straightforward and requires no new cryptography: website developers need simply define the issuers they will accept[13] and construct the access criteria they need using gadgets. For this scenario, the only issuer is our passport-based issuer, and the access criteria

[12] When augmented to hide the passport signature.

[13] This defines which credential list they use or defines a new list as some subset of existing ones.

	IssueReq	IssueGrant	ShowCred	ShowCred (full)	VerifyShow
Age-restricted vid.	1.97s	2ms	143ms	602ms	5ms
Entering a bar	1.97s	2ms	98ms	557ms	5ms

Table 3: `zk-creds` case study benchmarks. IssueReq is the time to convert a passport into a credential using zk-supporting-documentation. ShowCred is the time it takes a user to prove they are over 18. All other parameters are the same as in Table 1.

being proved are age, expiry, and non-cloning. Concretely, the access predicate is:

$$\mathsf{dob} \leq \mathsf{today} - 18\mathrm{yrs} \ \wedge \ \mathsf{expiry} > \mathsf{today} \wedge \ \mathsf{CloneResistance}(\mathsf{rk}, \mathsf{nonce}, \mathsf{tok}_1, \mathsf{tok}_2, \ldots)$$

where CloneResistance, nonce, tok_1, and tok_2 are as described in Section 5.3.

Table 3 gives performance numbers. Given a credential, it takes Alice 143ms to show a website she is over 18 (and 602ms when she must recompute her membership proof). The server can verify her assertion in 5ms. If we wish to optimize for server verification time or throughput, we can switch to `zk-cred`'s server-optimized construction and achieve 1.5ms verification times and 1.8 verifications per ms per core. Extrapolating from the server-optimized benchmarks in Table 1, proving times would increase to approximately 595ms.

Scenario 2: A cryptographer walks into a bar. To purchase alcohol in the United States, one needs to show photo ID and proof they are at least 21 years old. But showing a driver's license reveals name, sex, weight, and date of birth. And if the license's barcode is scanned [ACL12], additional information is revealed, potentially including whether the holder is insulin-dependent, hearing-impaired, developmentally disabled, or, surprisingly, a sexual predator [Veh].

We now build a system for in-person age verification coupled with photographic verification. Importantly, our goal in this scenario is *selective disclosure* and not anonymity. Anonymity in an in-person setting is often not possible or even desired. Rather, what we want is privacy: the ability to control what information is revealed and limit it to what is necessary—the patron's photo and the fact that they have an unexpired ID with a birth date making them of drinking age.

We envision a hypothetical system where bar patrons have an ID-wallet application on their phone. The app, using ShowCred, presents an identity assertion (e.g., via a QR code or NFC) to an app on a bouncer's phone which checks the assertion with VerifyShow and displays the user's photo along with whether they are over 21. In contrast to scanning the user's driver's license, this reveals only the minimal information necessary. The necessary access predicate is:

$$\mathsf{dob} \leq \mathsf{today} - 21\mathrm{yrs} \ \wedge \ \mathsf{expiry} > \mathsf{today} \ \wedge \ \mathsf{facehash} = H(\mathsf{face})$$

Table 3 gives concrete costs for local computations. To show that a patron is over 21 takes 98ms in the common case and 557ms if membership proofs must be recomputed. As before, verification is 5ms in either case.

8 Extensions and applications

We now discuss extensions to `zk-creds` and additional applications of our approach.

Other signed identity sources. As shown by our e-passport example, `zk-creds` can transform legacy identity sources into anonymous credentials if there is a digitally signed component. This raises an interesting question about what parts of existing identity and credential infrastructure include such signatures. For example, digital diplomas for many US universities include digital signatures over the diploma holder's name, degree, and institution. Many emails are signed with DKIM which, while problematic in many contexts [SPG21], could be a source of identity or membership in an organization. New York's Excelsior Pass for COVID vaccination contains the holder's name, birth date, and a signature.[14] Other existing digital protocols may contain a signature that establishes ownership of a resource (e.g., a phone number in an eSIM or virtual SIM card) or identity (e.g., Apple's digital driver's license features).

Complex access criteria. We have discussed conceptually simple access control criteria such as "my credential is not expired," or "I am of age," perhaps with a cryptographically complex mechanism for clone resistance. However, real-world access criteria can be far more complex. `zk-creds` provides a way to deal with such criteria without requiring coordination with identity issuing authorities for every custom access check that must be implemented.

An example of this comes in the form of online petitions and discussions. New York State has an online portal for discussion and petition which asks a user for their address to match them with the appropriate state senator [New]. While this check does not seem to be enforced, one could imagine both wanting to enforce this constituency check and allowing constituents to leave non-identifying comments. Similarly, some online resources, such as ebooks from the New York Public Library, are limited to city residents; enforcing this currently requires in-person registration for a library card to present proof of address, and opens up a (hypothetical) risk of tracking online reading habits [Ame06].

Geolocating an address to the bounds of, e.g., a city council district, however, is not simple. The computation is, by the standards of credential schemes, complex, and involves converting the address to a location and then perform-

[14] This was obtained by scanning the pass's QR code, whose contents are a W3C Verifiable Credential [W3C].

ing a point-in-polygon check.[15] For a small number of fixed boundaries (e.g., federal congressional districts), one might imagine avoiding the problem by issuing identity documents with this information included. But even in cases where the identity issuer would cooperate, coordinating all geocoding restrictions one might want to realize (e.g., anonymous discussion boards for a school zone, a neighborhood, or even specific apartment building) is impractical and may cause credential sizes to blow up.

Because `zk-creds` supports general purpose zero-knowledge proofs, geocoding restrictions are made more feasible with Groth16 gadgets: even if the Groth16 proof for the gadget is expensive, the resident or an outsourced prover avoids recomputing it every show. After the first time, the proof can be reused arbitrarily, until the user's Merkle tree is updated by a new issuance.

Sybil-resistant IDs from email or money. Internet services currently prevent multiple account registration (Sybils) by requiring the user to consume a (hopefully) scarce resource, such as money (via a micropayment), attention (via a CAPTCHA), and possession of, e.g., phone number or email address.

`zk-creds` provides several avenues for Sybil-resistant credentials with anonymity. Credentials can be issued based on signed identity documents (e.g., a passport, as demonstrated in Section 6) with the signature as a uniqueness check. Similarly, `zk-creds` can thwart Sybils via cryptocurrency: a simple smart contract issues credentials if and only if a small fee is paid.

Finally, and perhaps most surprisingly, we can use possession of a valid email address as a Sybil-resistance mechanism without the use of a trusted third party or cooperation with the email provider. A DomainKeys Identified Mail (DKIM) header, which appears on all outgoing mail of most modern email providers, contains a signature from the email provider. By embedding the credential issuance request in the email body, we get an externally verifiable proof of possession of an email address that can be used to issue Sybil-resistant accounts. This allows us to leverage the Sybil resistance mechanisms used by Gmail, for example.

These short- and long-lived IDs can be reused and rate-limited across actions, similar to the functionality of Privacy Pass [DGS+18], which issues one-time-use anonymous access tokens for every CAPTCHA a user completes.

Oracle- and self-issued credentials. A number of academic works and industrial systems have emerged to address the so-called "oracle problem": how does a consensus scheme such as a blockchain come to agreement about real world events?

One class of solutions [EJN] relies on incentive systems and the ability to challenge the veracity of data. These approaches, if viable, could be used to issue anonymous credentials based on public online reputation data (e.g., Twitter follower count, Reddit karma, etc.). Crucially, because `zk-creds` forces

[15] Indeed, geolocation is the correct way to prove residency. For example, the borders of New York City's 10 city council districts are defined by 1.8 megabytes of geospatial vector bounds [NYC].

all issued credentials to be on a public list, any malfeasance by an issuer could be detected and punished.

An orthogonal approach is the creation of notaries who attest to data on third party servers. DECO [ZMM+20] proposes a 2PC protocol between a client and a *notary* that authenticates data retrieved over a TLS connection from a third-party server. DECO also allows users to construct zero-knowledge proofs to only selectively disclose HTTP response contents. These features would allow a user to obtain a credential for their name and address by, e.g., logging into their utility provider and retrieving the bill. Moreover, in the event the user has trustworthy secure hardware, they can *self-issue* the credential by attesting the hardware ran this notary check itself.

We note, however, that such schemes inherently rely on assumptions (either non-collusion or the security of trusted hardware) that become increasingly infeasible as the value of the credential goes up. But, for example, it may be viable as a simple Sybil prevention or anti-spam mechanism.

Composable credentials. When new use cases for existing credentials emerge, they often require the combination of two different credentials. Take, for example, US-issued vaccine cards. Because these contain a person's name, but not a photo, COVID-19 vaccine mandates frequently required restaurants and bars to ask customers for a vaccine card *and* a photo ID with a matching name. `zk-creds` supports this type of post-hoc composition: two credentials both containing a field for, e.g., full name, can be jointly shown using the Join gadget described in Section 5.3.

9 Related work

Anonymous credentials derive from a long line of work, starting with Chaum [Cha85], and subsequently seeing numerous extensions [Cha85, CL01, CL03, CL04, CHK+06, BCKL08, CG08, BL13, GGM14, CDHK15, SAB+19]. While showing a credential initially allowed little more than (unlinkably) presenting a signed token connected to a user's pseudonym, the schemes were generalized and extended to provide more sophisticated properties such as issuance of hidden attributes, rate-limiting, k-show, and efficient selective disclosure of attributes. Because it uses general-purpose zero-knowledge proofs, `zk-creds` can (and does) capture all of these properties as implemented.

One drawback to deploying the majority of these schemes is the requirement of a single, trusted issuer. As such, existing work has sought to solve this issuance problem. We briefly compare and contrast other approaches to addressing this.

Distributed issuance. In 2014, Garman et al. [GGM14] were the first to propose the notion of decentralized anonymous credentials. Our approach is directly inspired by their work.

While the approach of Garman et al. removes the assumptions of a single issuer and the need for issuers to hold keys, it both leaves open a number of essential questions for operating a real system and introduces new ones which we address. First, showing a credential requires users to 1) locally store the full credential list and 2) compute proofs which take time linear in the number of issued credentials for every show (even for static credential lists). In contrast, `zk-creds` develops a new approach and new cryptography (blind Groth16) to allow proof reuse and fast credential showing. And, via the use of Merkle trees, `zk-creds` requires users to store only logarithmic-sized witness data to compute credential shows. Second, while Garman et al. suggest the possibility of more complex features, they do not implement them. More importantly, the set of bespoke sigma protocols they use does not provide for the composition of credentials and identity attributes or support complex identity statements. By developing a protocol based on general-purpose zkSNARKs, we do. Lastly and most significantly, the work does not answer the question of how the decision to issue a credential could be made without disclosing sensitive information to the issuer (i.e., the very problem zk-supporting-documentation addresses). This also makes it impossible to audit credentials that are issued via Garman et al.'s construction, in marked contrast to `zk-creds`.

Threshold issuance. The Coconut [SAB+19] anonymous credential scheme addresses the issuance challenge via threshold signatures. In Coconut, credentials are issued by n static parties under the assumption $t > 1/2$ of them are honest. The scheme is clever and achieves efficiencies on par with single-issuer schemes. However, while threshold issuance increases the security of a scheme by requiring an attacker to corrupt more parties, it only addresses the scarcity of issuers if we have an abundance of parties who are willing to issue credentials but, for whatever reason, no individual one is trusted. In many settings, it is a challenge to find even a single party who both 1) is empowered to make identity statements and 2) is willing and able to run cryptographic infrastructure even if, by fiat, we trust them. Moreover, Coconut only supports selective disclosure of attributes in a credential, not complex zero-knowledge proofs over attributes. It does not meet our design goals of flexibility or dynamic generation of access criteria.

Finally, we note that `zk-creds`, as shown by the version given in Table 2, supports credential issuance via threshold Schnorr schemes (e.g., FROST [KG21]), so we do capture the functionality of Coconut.

Decoupling issuance from identity verification. More broadly, another line of work, starting with TLSNotary [TLS], considers convincing third parties of the correctness of data. DECO [ZMM+20] extended this protocol to support TLS 1.3 and used zero-knowledge proofs for selective disclosure (e.g., the party learns a bank account balance is over a threshold, but not the balance itself or the account holder's identity). Applying this to the anonymous credential setting, one could use it to separate finding a cryptographic issuer for credentials

from the process of verifying entitlement to a credential. Such approaches are complementary to `zk-creds` and we consider them as an extension in Section 8. Without such integrations, however, such an approach would still require at least one trustworthy party to be willing to run a highly available web service that holds live signing keys.

Decentralized identity (DID). We note another area of research on decentralized identity which has been the subject of both academic and industrial work, including some standardization. While `zk-creds` is compatible with these goals inasmuch as it allows issuance without a party holding keys, such works are largely orthogonal: decentralized identity as an area considers who makes decentralized identity assertions, while our work considers how to transform some identity assertion (centralized or otherwise) into an anonymous credential without introducing additional trusted parties.

10 Conclusion and future work

The approach we develop here—a switch from blind signatures to zero-knowledge proofs as the foundation for anonymous credentials—implies several avenues for further work. In this section, we enumerate a few immediate consequences of this paradigm and future research areas.

Instantiating `zk-creds` with improved zero-knowledge proofs. We have instantiated `zk-creds` with Groth16. The `zk-creds` approach we develop, however, is proof system-agnostic. As such, instantiating the `zk-creds` paradigm with other proof systems, such as ones without trusted setup (e.g., [BBHR18, Gro16, BBB$^+$18]) or with universal setup (e.g., [MBKM19, GWC19, CHM$^+$20]), is entirely possible. For the monolithic construction, such a change is a drop-in replacement. If we wish to support precomputation of separate membership proofs, as in our client-optimized scheme, we must either adapt blind Groth16 to the new proof system or take an alternative approach, e.g., recursive proofs. The choice of proof system is also tied to the choice of accumulator scheme.

Instantiating `zk-creds` with alternative accumulator schemes or primitives. We have instantiated `zk-creds` using Merkle trees. However, as with proof schemes, the same approach generalizes to other accumulator mechanisms such as RSA accumulators [GGM14], polynomial commitments [KZG10], Verkle trees [Kus18], or perhaps special-purpose schemes for zkSNARKs [ZBK$^+$22]. Again, for many such accumulators, this is a simple black-box replacement of the membership zkSNARK. Instantiating `zk-creds` with alternative accumulators will offer different tradeoffs for witness size, witness update requirements, witness computation cost, and accumulator verification cost (and hence zkSNARK proving time). A particularly exciting prospect is the co-design of accumulator schemes and zkSNARKs to achieve drastically improved performance.

Co-design or co-selection of zero-knowledge proofs and credential schemes. Similarly, one could instantiate either our existing version of `zk-creds`, or a different construction, with different cryptographic primitives. For example, one might replace Poseidon with a newer, circuit-optimized hash function, perhaps making use of low-degree gates in proving systems like Plonk [GWC19].

Batched verification/verification in a smart contract via proof composition. One interesting question is how to reduce the cost of verifying many credential shows. Zebra [RPX+22] considers this problem for batch verifying classic blind signature based anonymous credentials in the context of smart contracts where computation is very costly. In Section 6.2, we give basic benchmarks using batched pairings for verification of our server-optimized construction. But it is also possible to construct a zero-knowledge proof that batches many shows into one proof that has sublinear (in the number of shows) and potentially constant verification cost. For example, SnarkPack [GMN22], which can dropped in to aggregate shows in our server optimized version, quotes a logarithmically scaling batching mechanism with verification costs of 8192 proofs in 163ms. The design of better aggregation mechanisms, potentially co-designed with a `zk-creds` style scheme, is an interesting avenue for future work.

Instantiating `zk-creds` with alternative (post-quantum) primitives. While we have instantiated `zk-creds` with primitives such as Groth16, as mentioned previously, the flexibility and modularity of our system easily allows for the swapping of any primitives with those that fulfill the necessary requirements. For example, one could easily build a post-quantum version of `zk-creds` by instantiating it with post-quantum secure primitives and zero-knowledge proofs such as [AHIV17, BBHR19].

11 Acknowledgements

We would like to thank Mary Maller for the idea and sketch of the LinkG16 proof system, Daniel Benarroch for starting the discussion of Merkle Forests, and Jeffrey Burdges for pointing out that passports contain signed credentials. Both were productive conversations at Real World Crypto 2019. We would like to thank the anonymous reviewers and shepherd for their helpful comments.

Michael Rosenberg's work was supported by the National Defense and Engineering Graduate (NDSEG) Fellowship. Jacob White and Christina Garman's work was partially supported by NSF grants CNS-1816422 and CNS-2047991. Ian Miers's work was supported in part by a Facebook Research Award. Image credits [Lit].

References

[ACL12] ACLU Blog - Jay Stanley. A Creeping Private-Sector "Checkpoint

Society"—and a Small Step to Protect Your Privacy, 2012. https://www.aclu.org/blog/national-security/creeping-private-sector-checkpoint-society-and-small-step-protect-your.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[Ame06] American Civil Liberties Union. Librarians Speak Out for First Time After Being Gagged by Patriot Act, 2006. https://www.aclu.org/press-releases/librarians-speak-out-first-time-after-being-gagged-patriot-act.

[Ar22] Arkworks-rs. *Arkworks Ecosystem Homepage*, 2022. https://arkworks.rs/.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In IEEE S&P 2018 [IEE18], pages 315–334.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 132–145. ACM Press, October 2004.

[BCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security*

and Privacy, pages 287–304. IEEE Computer Society Press, May 2015.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, March 2008.

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

[BKSV21] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. In *Financial Cryptography and Data Security*, 2021.

[BL09] Ernie Brickell and Jiangtao Li. Enhanced privacy id from bilinear pairing. Cryptology ePrint Archive, Report 2009/095, 2009. `https://ia.cr/2009/095`.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013.

[BMM+19] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177, 2019. `https://eprint.iacr.org/2019/1177`.

[Bow17a] Sean Bowe. *Bellman: zk-SNARKs in Rust*, 2017. `https://electriccoin.co/blog/bellman-zksnarks-in-rust/`.

[Bow17b] Sean Bowe. *BLS12-381: New zk-SNARK Elliptic Curve Construction*, 2017. `https://electriccoin.co/blog/new-snark-curve/`.

[But] Vitalik Buterin. An Incomplete Guide to Rollups.

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 262–288. Springer, Heidelberg, November / December 2015.

[CG08] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 345–356. ACM Press, October 2008.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 1985.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 201–210. ACM Press, October / November 2006.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zk-SNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[CL03] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 268–289. Springer, Heidelberg, September 2003.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1445–1459. ACM Press, November 2020.

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

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Internet Activities Board, May 2008.

[CV02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi

Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002.

[DFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254. IEEE Computer Society Press, May 2016.

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

[DMMK18] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. In IEEE S&P 2018 [IEE18], pages 108–126.

[e-E] e-Estonia. ID-card. `https://e-estonia.com/solutions/e-identity/id-card/`.

[EJN] Steve Ellis, Ari Juels, and Sergey Nazarov. Chainlink: A decentralized oracle network. `https://research.chain.link/whitepaper-v1.pdf`.

[Eth] Ethereum Development Docs. Optimistic rollups. `https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/`.

[Fed] Federal Office for Information Security. German eID. `https://www.bsi.bund.de/EN/Topics/ElectrIDDocuments/German-eID/german-eID.html`.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018.

[GGM14] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *NDSS 2014*. The Internet Society, February 2014.

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

[GMN22] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical snark aggregation. In Ittay Eyal and Juan Garay, editors, *Financial Cryptography and Data Security*, pages 203–229, Cham, 2022. Springer International Publishing.

[Gooa] Google. Access age-restricted content & features - Google Account Help. `https://support.google.com/accounts/answer/10071085?p=age-verify`.

[Goob] Google. Watch age-restricted videos. `https://support.google.com/youtube/answer/10070779`.

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

[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. `https://eprint.iacr.org/2019/953`.

[ICA] ICAO. Machine Readable Travel Documents. Part 10: Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC). Technical report, International Civil Aviation Organization (ICAO). `https://www.icao.int/publications/Documents/9303_p10_cons_en.pdf`.

[IDN] IDNYC. About IDNYC. `https://www1.nyc.gov/site/idnyc/about/about.page`.

[IEE18] *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018.

[KG21] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. In *Selected Areas in Cryptography*, 2021.

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. Cryptology ePrint Archive, Report 2021/219, 2021. `https://eprint.iacr.org/2021/219`.

[Kus18] John Kuszmaul. Verkle trees. Technical report, MIT, 2018.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[KZM+16] Ahmed E. Kosba, Zhichao Zhao, Andrew K. Miller, Yi Qian, T-H. Hubert Chan, Charalampos Papamanthou, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. 2016. https://eprint.iacr.org/2015/1093.

[Lit] Litviniuk, Anna. Supportfemale 2 SVG. CC Attribution License https://creativecommons.org/licenses/by/3.0/.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[MTBI+22] Jose L. Muñoz-Tapia, Marta Belles, Miguel Isabel, Albert Rubio, and Jordi Baylina. CIRCOM: A Robust and Scalable Language for Building Complex Zero-Knowledge Circuits. 2022. https://www.techrxiv.org/articles/preprint/CIRCOM_A_Robust_and_Scalable_Language_for_Building_Complex_Zero-Knowledge_Circuits/19374986.

[New] New York State Senate. Find your senator. https://www.nysenate.gov/registration/nojs/form/start/find-my-senator.

[NYC] NYC Planning. Political and Administrative Districts — Download and Metadata. https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page.

[Rea] ReadID. Which countries have ePassports? https://www.readid.com/blog/countries-epassports.

[RPX+22] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. Cryptology ePrint Archive, Paper 2022/1286, 2022. https://eprint.iacr.org/2022/1286.

[SAB+19] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *NDSS 2019*. The Internet Society, February 2019.

[SPG21] Michael A. Specter, Sunoo Park, and Matthew Green. KeyForge: Non-attributable email from forward-forgeable signatures. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1755–1773. USENIX Association, August 2021.

[TLS] TLSNotary. Prove an HTTPS page was in your browser. https://tlsnotary.org/.

[Tora] Tor Blog. "One cell is enough to break Tor's anonymity". https://blog.torproject.org/one-cell-enough-break-tors-anonymity/.

[Torb] Tornado Cash. Core deposit circuit. https://docs.tornado.cash/tornado-cash-classic/circuits/core-deposit-circuit.

[U.S] U.S. Department of State, Bureau of Consular Affairs. Passport Services: Processing Times. https://travel.state.gov/content/travel/en/passports/how-apply/processing-times.html.

[Veh] Florida Highway Safety and Motor Vehicles. 2D barcode reader calibration sheet — 2016 AAMVA standard. https://www.flhsmv.gov/pdf/newdl/fl_2dbarcodecalibration.pdf.

[W3C] W3C Recommendation. Verifiable Credentials Data Model v1.1. https://www.w3.org/TR/2022/REC-vc-data-model-20220303/.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. Cryptology ePrint Archive, Paper 2022/621, 2022. https://eprint.iacr.org/2022/621.

[ZMM+20] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. Deco: Liberating web data using decentralized oracles for tls. In *ACM CCS 2020*, 2020.

A Performance across parameter choices

We expand on the microbenchmarks in Section 6.2 by investigating how `zk-creds` scales with respect to various parameters.

Recall that showing a credential requires proving: 1) the credential is in the list of issued credentials, 2) the relevant attributes are part of the credential, 3) the attributes meet the access criteria being shown, 4) each of the above is about the same data (linkage proof). Thus, the performance of `zk-creds` depends on the number of issued credentials (via 1 above), the size of the attribute (via 2), and the complexity of the access criteria (via 3).

Membership benchmarks. Recall that a membership proof consists of a proof of credential membership in a tree, followed by a proof of membership of that tree in a forest. In Figure 6, we show the performance of proving membership as the shape of the forest changes. For a fixed number of total leaves, we find the size of the forest (and, consequently, height of its trees) that minimizes

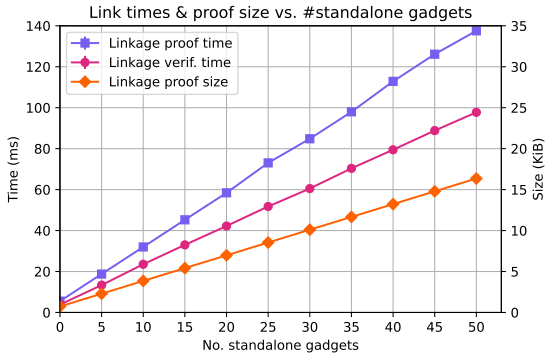

Figure 5: Costs of linkage proofs as #standalone gadgets varies.

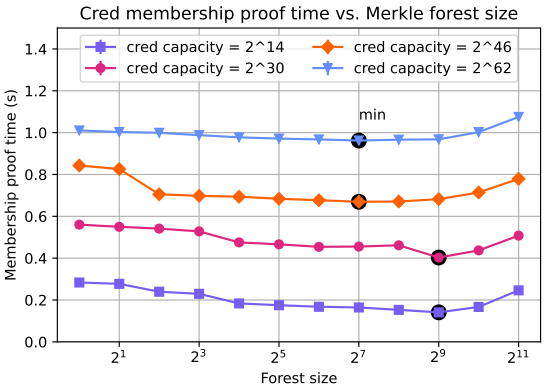

Figure 6: Costs of proving tree then forest membership, as number of trees in forest varies.

#leaves	2^{14}	2^{30}	2^{46}	2^{62}
Issuance proof time	141ms	403ms	670ms	962ms

Table 4: Cost of a proof of credential list membership as the size of the list varies.

membership proving time. This results in a 50% (143ms) speedup in the best case. Further, the verifier pays nothing for this optimization, since all public inputs are prepared in advance and reused for all verifications.

In Table 4, we show the time to compute a proof of membership in the credential list as the list size varies. This represents the baseline cost of the issuance portion of any ShowCred call. The benchmark consists of one Groth16 proof of tree membership plus one Groth16 proof of forest membership. For a fixed number of leaves, the tree height is chosen using the optimal parameters from the experiments in Figure 6.

Linkage benchmarks. In Figure 5, we plot the size, proving time, and verification time of linkage proofs as the number of standalone gadget proofs varies. Every additional gadget adds 330B to the proof size.

B Merkle forests and witness management

Showing a credential requires proving membership in a Merkle tree. Since the list of issued credentials is dynamic, users must somehow maintain an up-to-date witness to their credential's membership in the tree. A naïve approach requires $O(N)$ storage[16] and $O(N)$ communication: clients store the entire tree and retrieve every addition. In this section, we attempt to solve the issues

[16]This is trivially reduced to $O(log(N))$*storage* per credential, as the client can update witnesses in place.

introduced in Section 5.1, iteratively building up techniques to optimize Merkle trees and their membership proofs in the interests of both efficiency and privacy. First, we address common problems with using Merkle forests which motivate this discussion and clarify the assumptions our constructions make. Second, in Appendix B.2 we look at various interoperable batching techniques for Merkle trees, allowing issuers to summarize *prior* Merkle tree witness updates for specific clients based on when they last received an updateand/or defer *future* credential issuance until many credentials can be added in a single batch to the Merkle tree(s). Third, in Appendix B.3 we look at ways to minimize correlation attacks against privacy due to users requesting then immediately showing their credential's membership proof, and suggest replacing update requests with a universal broadcast of all relevant updates.

We conclude our discussion of Merkle tree optimizations with a unified Merkle forest construction in Appendix B.4 which balances the competing tradeoffs of 1) expected number of membership proof updates, 2) privacy, and 3) communication overhead. We accomplish this by combining the aforementioned techniques into a unified approach which leverages the unique structure of Merkle trees and authentication paths. Concretely we find that, compared to a single Merkle tree of similar capacity, our construction requires fewer membership proof updates over time for *all* users of older credentials, at the expense of only a small-constant logarithmic increase in communication overhead between issuers and users.

B.1 Merkle Trees in Practice: Overview and Motivations

In `zk-creds`, users must frequently produce membership proofs across the lifetime of a credential to show that it has been issued on a credential list. Doing so requires a user to obtain some new information about the list—the *frontier*—with which users can update the witness to their credential's issuance.

A common approach to allow proving membership on a list is to instantiate it as as a full binary Merkle tree with capacity N, allowing anyone with the credential and its $O(\log N)$-sized witness (i.e., the *authentication path* θ) to verify that it is a leaf node included in the computation of the list's Merkle root. Naïve Merkle tree-based instantiations might require the user to manage a local copy of the entire Merkle tree themselves, or otherwise delegate the task of updating and distributing the list to some *list manager* who is only trusted to not perform DoS attacks against the system. In real systems, however, keeping up-to-date with the current state of an entire Merkle tree is a non-trivial problem: credential lists can be fast-growing or large; users' devices may be particularly resource-constrained and thus unable to process or even store the entire list at once (e.g., envisoning its usage on a mobile phone or domain controller);[17] users who become inactive for long periods of time will

[17]For reference, storing a full 32-height Merkle tree with 256-bit hashes requires about 137 GB.

need to request much of the updated list; and users who miss any part of the update would struggle to return to a consistent state.

Moreover, since `zk-creds` is designed such that an honest user's credential remains unlinkable even when a dishonest majority of malicious parties collude, our credential list instantiation should support strong unlinkability and privacy in practice as well. However, we observe that a user updating an authentication path through *any* kind of request to a third-party will correlate closely with the subsequent usage of that credential, since doing so is necessary to maintain a valid membership proof. This leaves users vulnerable to correlation attacks regardless of whether the updates sent back to the user identify the credential.[18] Correlation attacks especially pose a problem if collusion occurs between an issuer and verifier or if the authentication path update identifies or reduces the anonymity set of the requesting credential. If adversaries can build plausible profiles to identify credentials within and across various shows in this manner, this has serious implications on unlinkability and user privacy. We will address these various problems in the remainder of this section.

Hereafter, assume that issued credentials can only be added to the Merkle tree; list entries will never change in-place. For simplicity, assume that credentials are only added from left to right. Lastly, assume that Appendix B.2–B.3 uses the conventional construction for Merkle forests, where each Merkle tree in the forest has fixed height n and capacity $N := 2^n$; see Appendix B.4 for design decisions and optimizations that can arise when these assumptions are relaxed.

B.2 Batching issued credentials

Requesting summaries of newly issued credentials. Observe that, if a user has a valid Merkle authentication path (i.e., witness) θ attesting to their credential's issuance at time t, not all $n+1$ nodes in θ will usually need updating by time $t' > t$. Instead, the user only needs a *summary* of all $\leq n+1$ Merkle tree nodes which have been added since time t. While users still reveal identifying information about the credential when updating their authentication paths with this approach—reducing the anonymity set to anywhere from $N/2$ to 1 credentials depending on the request—this approach is somewhat better at both mitigating privacy risks and reducing communication overhead as well compared to the naïve approach of requesting the entire θ each time.

Deferring credentials to issue. One of the more effective techniques to make credential list updates more efficient is to allow the issuer to update the

[18] Practical session privacy protections are also insufficient. Even anonymous communication networks such as Tor are (perhaps inherently) vulnerable to de-anonymization via various correlation attacks [DMMK18, Tora].

credential list in *batches* according to a specified epoch. Instead of immediately adding a validated cred to a Merkle tree T (as with IssueGrant in Figure 4), the issuer could, for example, define a regular interval Δ_I to wait before issuing all valid credentials they collected at once. For example, if Alice's credential is being issued in this manner, she must wait at most Δ_I seconds before she can present a valid membership proof for her issued credential.[19] After the batched credentials are added, Alice can obtain its initial witness as appropriate. Note that an issuer can decide on an appropriate batching policy based on usability and efficiency tradeoffs.

Synchronizing Merkle tree state. More generally, we could consider some agreed-upon list manager (perhaps distinct from the issuer) who determines the current state of the Merkle tree for the purposes of membership verification. Even if the issuer themselves is unwilling or unable to batch in the manner described above, the list manager can *snapshot* the state of the Merkle forest according to a specified epoch, e.g. at a regular interval Δ_ℓ, and distribute this snapshot as the "current" agreed-upon state of the Merkle forest.

For example, suppose that the latest snapshot occurred at time t and that snapshots are published every Δ_ℓ seconds. First, the user updates their authentication path θ to be valid as of time t using the latest snapshot. If the user and verifier agree that the latest snapshot is treated as current, then θ is automatically up-to-date; otherwise, the user now only needs to manage updates to θ between times t and $t + \Delta_\ell$.

In either case, this approach can be particularly useful for users who have been offline and unable to update their authentication path for many epochs, or who have corrupted their local copy and need to recover the state of the credential list; snapshots of the credential list allow the user to more easily remain up-to-date.

These techniques for batching issued credentials are mostly agnostic to the list's instantiation. As we will soon see, however, this technique is amenable to other approaches where Merkle trees are used.

B.3 Broadcasting frontier nodes

Instead of the user *requesting* specific updates to frontier nodes in θ, as in Appendix B.2, consider an alternate approach where a list manager proactively *broadcasts* the relevant frontier update(s) to all users. Each addition to the Merkle tree requires the list manager to broadcast at most $1 + \log_2 N$ nodes to all users, i.e. all updated nodes (as relevant to existing credentials) that changed after adding one or more credentials. If left unbatched, this requires

[19] Indeed, many real-world credentials already require a waiting period. For example, a US passport can take 5–11 weeks to validate and 1–2 days to ship before it can be used [U.S].

broadcasting a small-constant $O(N \log N)$ number of nodes to each user across the lifetime of the system.

However, one can reduce the number of nodes to stream by summarizing a sequence of newly issued credentials. As a simple example, if an issuer issued 2^b credentials by placing them in a binary Merkle tree, it suffices to broadcast a single path of at most $n - b + 1$ nodes from the updated root of the tree to the subroot of *all* new batched credentials. With this, the user of any previously-issued credential can parse any updated nodes they need for their authentication path θ to be current up to that batch. Since broadcasted tree updates now only occur once per batch instead of once per credential, this helps decrease communication overhead by a factor of at least 2^b.

Together, summarizing batches of issued credentials and broadcasting frontiers preserves the instantiation's unlinkability across credential usages. Notably, users no longer need to send identifying information about their credential to update its authentication path, since the broadcasted frontier nodes contain every relevant update to the Merkle tree for all users. Furthermore, proactively broadcasting the frontier updates to users significantly reduces the ability for adversaries to correlate credential usage, as users no longer even need to reveal via request when they want to update their authentication path.

While this approach comes at the cost of additional overhead and system complexity compared to the approach in Appendix B.2, it is especially suitable for applications where credential lists are relatively static and credentials do not need to be issued immediately. This approach is also better for many users compared to the naïve approach of downloading the entire Merkle tree, especially when using resource-constrained devices. Similar to standard Merkle-based constructions, the user only needs to store a logarithmic number of nodes (roughly $\log_2 N + c$, where c is a small constant) to process the stream of frontier nodes and update θ. Furthermore, while broadcasting all Merkle tree updates results in $O(N \log N)$ frontier nodes (more precisely, at most $\sum_{i=1}^{N} 1 + \log_2 N = N \cdot (n + 1)$) being sent to each user across the lifetime of the Merkle tree, the stream can be processed periodically and in only a single pass using a logarithmic-sized buffer.

B.4 Optimizing Merkle forests

In this subsection, we propose a novel construction which leverages many of the techniques discussed above to simultaneously minimize witness update frequency(and thus proof computation costs), privacy risks (via update request interactivity), and user storage costs at the expense of a moderate increase in communication overhead. We believe that this provides an amenable tradeoff between privacy, efficiency, and usability for many use-cases leveraging Merkle trees. In particular, users can (eventually) rely solely on broadcasted frontier

nodes to update their authentication path θ, the Merkle forest scales effectively as the number of credentials grows, and every user will be required to update their credentials' proofs of membership less often compared to a naïve Merkle tree construction of comparable capacity.

Consider an alternate Merkle forest instantiation $F := (r_1, r_2, \ldots, r_n)$ where each Merkle root r_i corresponds to a full binary Merkle tree T_i *with height* i, and where n is the current number of roots in F. As such, we define the capacity of F as $N := \sum_{i=1}^{n} 2^i = 2^{n+1} - 2$.

The Merkle forest approach below relies on two crucial observations. First, *any* approach based on balanced Merkle forests will have smaller tree heights than a single Merkle tree of comparable capacity (a single Merkle tree with capacity 2^{n+1} has height $n+1$). Second, we can leverage the fact that, with multiple Merkle trees, the authentication path of a credential in tree T_i will only change when other credentials are added to that *same* tree T_i. Modifications to any other tree $T_j \neq T_i$ in the forest F will only invalidate the user's forest-membership proof (which is a simple OR proof) but not their tree-membership proof, allowing reuse of the latter proof via LinkG16.

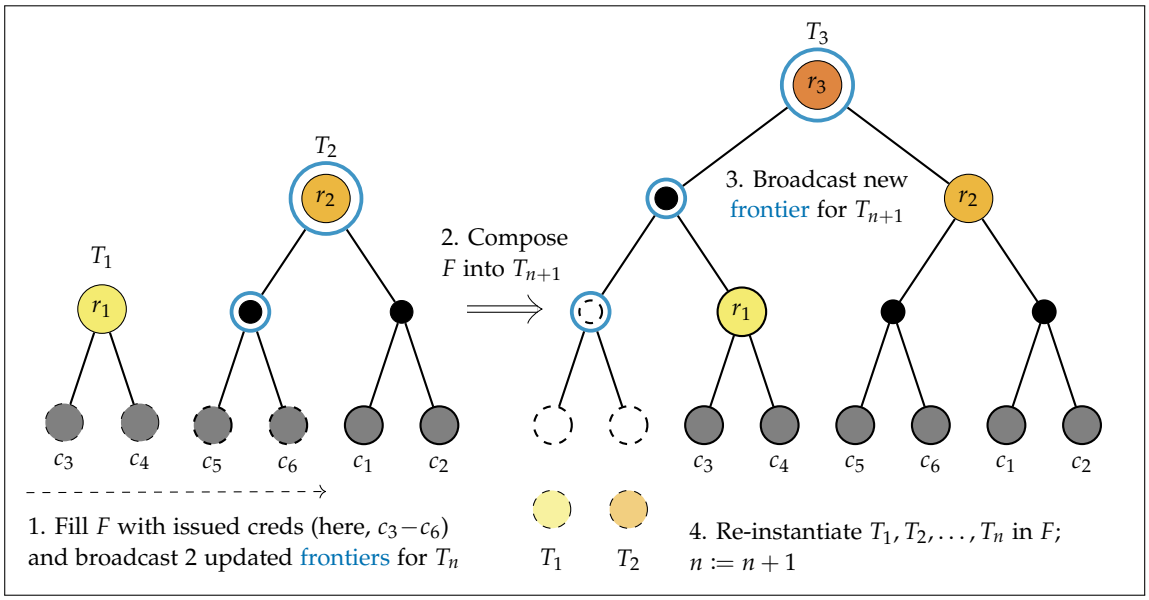

Figure 7: A single iteration of the forest-packing algorithm with $n_0 = 1$ trees in F at initialization and $n = 2$ trees as of Step 1. Dashed nodes represent hashes of empty entries or a root of an empty subtree; grey leaf nodes are hashed entries (i.e., issued credentials); blue-circled nodes are frontier nodes. Note that, invariably, the forest with n_0 trees requires no frontier because of Step 0 in the witness-update algorithm.

Growing Merkle forests. Assume without loss of generality that the user knows the representation of the broadcasted frontier updates, and (for sim-

plicity of analysis alone) that issuance and authentication path updates are left unbatched. We also assume, as before, that credentials are not modified in-place once added. We first define a *forest-packing algorithm* describing how a list manager or issuer adds newly issued credentials to Merkle forest F:

0. Initialize F with $n_0 > 0$ sparse binary Merkle trees, where each tree T_i has height i and all leaves/entries are currently empty (i.e., the hash of nil). That is, F initially contains $n \coloneqq n_0$ empty trees.
1. Populate the entries in F with newly issued credentials as desired (e.g., from left to right, starting with the smallest available tree) until F is full.
2. Compose all now-filled trees $T_1, \ldots, T_n$ into a single tree T_{n+1} with height $n+1$, arranging each T_i into a sequence of right subtrees. Initialize the remaining 2 leftmost entries in T_{n+1} as empty and add r_{n+1} to F.
3. Immediately, and after any future addition to the 2 leftmost entries from Step 1, broadcast the *frontier* to T_{n+1} containing its "leftmost" intermediate nodes and the root r_{n+1} that updated.
4. Re-instantiate $T_1, \ldots, T_n$ and their corresponding roots in F as empty, then repeat from Step 1 with $n \coloneqq n+1$.

Updating authentication paths. For clarity, we also define a *witness-update algorithm* which describes how a user tracks changes to their issued credential's authentication path θ as F grows. Suppose that the user can only reliably track the current *roots* of each Merkle tree F, and does so out-of-band.[20] Then:

0. The user first requests to issue a credential. If valid, the issuer adds it to some Merkle tree T_i in F, allowing the user to obtain their initial witness/authentication path θ. *Initially*, a user must track updates to θ as appropriate until T_i is full.
1. Once the tree T_i holding the credential is filled, the user waits until all trees in F are completely filled; only the roots in $F \setminus \{T_i\}$ may change after this point, so θ remains a valid Merkle path to the credential.
2. The user listens for a broadcast indicating that their credential has been composed into the new tree T_{n+1} (see above). They scan in a single pass for the relevant frontier node(s) which, in conjunction with the old θ and (now sub-)roots in F, form the new authentication path for T_{n+1}; the user updates θ accordingly and marks that F has grown with $n \coloneqq n+1$.
3. Thereafter, their credential's tree will invariably have 2 empty entries; listen for broadcast(s) indicating that an new credential changed the relevant frontier nodes, and update θ accordingly. Repeat from Step 1.

[20] Even for massive Merkle forests of capacity, say, $2^{33} - 2 = 8.6$ billion credentials, this only requires tracking and storing 32 hash values (i.e. for 256-bit digests, 1 KB).

To see how a user (say, Alice) can correctly compute her current authentication path θ in this construction, consider the lifetime of her issued credential. Without loss of generality, consider her credential to be initially committed to by the i-th Merkle tree T_i (of n) in the Merkle forest F. Alice is initially responsible for updating or requesting θ herself, doing so until the tree T_i that her credential resides in is full. Thereafter, her credential will be composed into the largest tree of the forest, and after this point it suffices for Alice to track changes to θ using frontier updates alone to keep her membership proof updated. More precisely, any user with a credential in the (now sub-)trees $T_1, \ldots, T_n$ can update its authentication path θ to be current for the tree of height $n+1$ by scanning for the frontier nodes in the Merkle path which compute from the subroot (i.e., the old r_i) up to the current Merkle root r_{n+1}, also considering the roots of the subtrees that were just composed into T_{n+1}. By repeating this process for updates to this largest tree (invariantly, at most two more times), it follows by induction on the number of trees in the Merkle forest F that the user can correctly update to a valid authentication path θ across the lifetime of the credential, and can do so by using only broadcasted frontier nodes and the forest's roots after the initial tree is filled.

Comparison to Merkle tree. One of the primary benefits of this Merkle forest approach is that it allows users to reuse their authentication path more often than the approaches using fixed-height Merkle trees, greatly reducing how often a user must recompute expensive tree membership proofs.

For all users of older credentials which have already been composed into the largest tree in F, each user will only have to update θ at most three times per new tree. As such, this only requires users of older credentials to perform $O(\log N)$ updates (excluding updates while in the initial tree; see below) for the remaining lifetime of the Merkle forest. Furthermore, broadcasting a global frontier eliminates the need for users of older credentials to interact with other parties to update authentication paths, providing strong privacy guarantees beyond the short-term.

Even for users of newly issued credentials whose credentials are still in their initial tree T_i, this Merkle forest approach requires fewer authentication path updates compared to a standard Merkle tree of near-identical capacity (i.e. $T_1, \ldots, T_n$ vs. T_{n+1}). In the worst case, where a credential is initially added to tree T_n, this requires a credential to update its authentication path at most 2^n times as other added credentials update its Merkle root (compared to 2^{n+1} for a standard Merkle tree). However, depending on how the issuer distributes the newly issued credentials across the variable-height trees in F, users might expect a reduction from the worst case of 2^n initial updates. For example, assuming an average case where a credential has a uniform probability of being added to any given leaf node (i.e., average number of updates for each tree weighted by tree capacity in the forest), in expectation a user would need to perform $\sum_{i=1}^{n} \frac{2^i}{N} \cdot 2^{i-1} = \frac{1}{3}(2^n + 1)$authentication path updates in the initial

tree. As such, the Merkle forest approach requires 2–3× fewer initial updates (before batching) compared to a single Merkle tree of near-identical size.

And, while the reduced number of updates comes at the expense of increased communication costs to broadcast to all users, the tradeoff is still competitive compared to standard Merkle trees in some cases. Since at most three frontiers will be broadcasted for each additional tree T_{i+1} added to a Merkle forest F as it grows from n_0 to $n \geq n_0$ trees, and each frontier contains i nodes, the system must broadcast at most $\sum_{i=n_0+1}^{n} 3i = \frac{3}{2}(n - n_0)(n + n_0 + 1)$ frontier nodes to each user across its lifetime. Concretely, assuming 256-bit hash digests and that the Merkle trees in the forest start with height $n_0 = 1$ and end with height $n := 32$ (i.e., 8.6 billion credentials), processing the frontier would require broadcasting at most 51 KB to each user across the lifetime of the system.

Overall, the issuer or list manager must broadcast a small-constant $O(\log^2 N)$ number of nodes to each user, compared to sending $O(\log N)$ nodes per user request as with the naïve Merkle tree approach. Furthermore, since only 3 broadcasts are required per tree with increasing height i, the rate at which broadcasts occur will decrease exponentially as F grows.

Distributing newly issued credentials. We emphasize that, if using the Merkle forest packing approach, the issuer or list manager has power to add credentials in an arbitrary manner to any tree $T_1, \ldots, T_n$ of variable height in F by default. This asymmetry in initial tree sizes has important implications on computational fairness and efficiency for users wishing to mitigate proof re-computations for their newly issued credentials, even outside an adversarial setting. We leave further discussion on the implications and importance of asymmetry in Merkle forests to future work.

C Revocation of issued credentials

In certain situations, some party might wish to revoke an already-issued credential, invalidating its authentication path θ using mechanisms beyond simple checks for, e.g., expiry. A naïve approach for credential revocation would be to replace its leaf node in the Merkle tree with a hash of nil, as was done with RevokeCred for fixed-height Merkle forests in Figure 4. However, in-place revocation such as this comes with significant drawbacks. For example, naïve revocation would cause many of the optimizations mentioned before (especially batched requests and the forest-packing algorithm) to longer work. However, other revocation mechanisms for Merkle trees may allow `zk-creds` to support both revocation and the Merkle tree optimizations described above.

On the surface, anonymous credentials and X.509 certificate often employ similar methods for managing membership on a list. As such, one might be tempted to use the idea of certificate revocation lists (CRLs), as defined in RFC 5280 [CSF+08], for *anonymous* credentials as well. This would avoid many

issues with deleting the credentials in-place on the membership list. However, there are important consequences to using CRL-based methods for revocation.

Perhaps most importantly, for revocation to be possible RFC 5280 requires certificates to have some form of persistent identifier; this is not possible for all types of anonymous credentials, however. More precisely, a CRL-compatible anonymous credential must contain a unique, persistent, and privacy-preserving identifier id (e.g., a credential hash or committed attribute). Additionally, either the user must specifically request for their credential to be revoked, or a verifier/auditor must be able to de-anonymize a misused credential (e.g. through violation of cloning resistance) to revoke it.

A CRL-based revocation approach for anonymous credentials would likely also come with many of the same practical issues as with CRLs for certificates. For example, strict granularity and consistency of updates is required for all parties to maintain a secure and synchronized revocation list. Additionally, trusting a party to manage a credential revocation list may re-introduce trust assumptions that some central authorities' signing keys are kept secret and are used honestly (see [CSF+08], Section 3.3). We leave a more thorough investigation of `zk-creds`-compatible revocation techniques to future work.

D Zero-knowledge definitions

To prove security of the protocols in Appendix E and Appendix F, we require definitions of soundness and zero-knowledge for proof systems. The definitions below are identical to those of [BMM+19]. $\langle \mathsf{A}, \mathsf{B} \rangle$ denotes the transcript of a protocol run between algorithms A and B.

Definition (Knowledge-sound argument)**.** *A public-coin argument* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *on a relation* R *is knowledge-sound with error* $\kappa(\lambda)$ *iff for all deterministic efficient (possibly dishonest) provers* P^**, there exists an efficient extractor* E *such that for all PPT adversaries* A*,*

$$\Pr\left[\mathsf{tr} \text{ accepts} \wedge (x,w) \notin R \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \mathsf{tr}) \leftarrow \langle \mathsf{P}^*, \mathsf{Verify} \rangle (\mathsf{crs}) \\ w \leftarrow \mathsf{E}^{P^*}(\mathsf{crs}, x, \mathsf{tr}) \end{array}\right] \leq \kappa(\lambda).$$

We say Π *is knowledge-sound iff it is knowledge-sound with negligible error.*

Definition (Perfect honest-verifier zero-knowledge (HVZK))**.** *Let* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be an interactive argument of knowledge on a relation* R*. Let an adversary be a pair of PPT algorithms* $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$ *such that* $\mathsf{A}_0(\mathsf{crs})$ *picks an instance, witness, and random coins* (x, w, ρ)*; and* $\mathsf{A}_1(\mathsf{tr})$ *decides whether a transcript is the result of a simulation or not.*

Π *is perfect honest-verifier zero-knowledge iff there exists an efficient simulator* Sim *such that for all adversaries* $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$*,*

$$\Pr\left[(x,w) \in R \wedge \mathsf{A}_1(\mathsf{tr}) \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,w,\rho) \leftarrow \mathsf{A}_0(\mathsf{crs}) \\ \mathsf{tr} \leftarrow \mathsf{Sim}(\mathsf{crs}, x\,;\,\rho) \end{array}\right]$$

$$= \Pr\left[(x,w) \in R \wedge \mathsf{A}_1(\mathsf{tr}) \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,w,\rho) \leftarrow \mathsf{A}_0(\mathsf{crs}) \\ \mathsf{tr} \leftarrow \left\langle \begin{array}{l} \mathsf{Prove}(\mathsf{crs}, x, w), \\ \mathsf{Verify}(\mathsf{crs}, x\,;\,\rho) \end{array} \right\rangle \end{array}\right]$$

E LinkG16

We now describe and prove the security of the LinkG16 proof system.

E.1 Goal

The purpose of LinkG16 is to show that k Groth16 proofs over heterogeneous circuits $\mathsf{crs}_1, \ldots, \mathsf{crs}_k$ all share the same first t public inputs $\{a_0, \ldots, a_{t-1}\}$ *without* revealing the inputs. That is, given k Groth16 proofs $\pi_1, \ldots, \pi_k$, we wish to construct a zero-knowledge proof of the following relation:

$$R_{\mathsf{linkg16}} = \left\{ \begin{array}{c} (\{\mathsf{crs}_i, \hat{S}_i\}_{i=1}^k; \{a_j\}_{j=0}^{t-1}, \{\pi_i\}_{i=1}^k) : \\ \bigwedge_{i=1}^{k} \mathsf{G16.Verify}(\mathsf{crs}_i, \pi_i, \hat{S}_i + \sum_{j=0}^{t-1} a_j W_j^{(i)}) \end{array} \right\}$$

where $W_j^{(i)}$ represents the wire W_j in crs_i, and $\hat{S}_i = \sum_{j=t}^{\ell} a_j W_j^{(i)}$ is the verifier-known prepared input for the i-th proof.

E.2 Construction

We define the new proof system below.

$\mathsf{LinkG16.Link}(\{a_j\}_{j=0}^{t-1}, \{\mathsf{crs}_i, \pi_i\}_{i=1}^k) \to \pi_{\mathsf{link}}$ Sample values $z_1, \ldots, z_k \leftarrow \mathbb{F}$ for blinding. For each i, commit to the shared inputs, $U_i := z_i[\delta]_1^{(i)} + \sum_{j=0}^{t-1} a_j W_j^{(i)}$. Let π_{eqwire} be an EqWire discrete-log equality proof (described in Appendix F) that the U_i commit to the same a_j values,

$$R_{\mathsf{eqwire}} = \left\{ \begin{array}{c} (\{U_i, \mathsf{crs}_i\}_{i=1}^k; \{a_j\}_{j=0}^{t-1}, \{z_i\}_{i=1}^k) : \\ \bigwedge_{i=1}^{k} U_i = z_i[\delta]_1^{(i)} + \sum_{j=0}^{t-1} a_j W_j^{(i)} \end{array} \right\}.$$

Rerandomize the underlying proofs in place, $\pi_i \coloneqq \mathsf{G16.Rerand}(\mathsf{crs}_i, \pi_i)$, then blind the proofs,

$$\pi_i' \coloneqq (A_i, B_i, C_i - [z_i]_1).$$

The final output is

$$\pi_{\mathsf{link}} \coloneqq (\pi_{\mathsf{eqwire}}, \{U_i, \pi_i'\}_{i=1}^k).$$

$\mathsf{LinkG16.LinkVerify}(\pi_{\mathsf{link}}, \{\mathsf{crs}_i, \hat{S}_i\}_{i=1}^k) \to \{0,1\}$ Check π_{eqwire} using $\mathsf{EqWire.Verify}$. Then unpack each π_i' into (A_i', B_i', C_i'). For each $i = 1, \ldots, k$, check

$$e(A_i', B_i') \stackrel{?}{=} e([\alpha]_1^{(i)}, [\beta]_2^{(i)}) \cdot e(C_i', [\delta]_2^{(i)}) \cdot e(U_i + \hat{S}_i, H).$$

where $\hat{S}_i$ is the Groth16 prepared public input for circuit i.

E.3 Proofs

Theorem 1 (Correctness)**.** *If* $\mathsf{G16.Prove}$ *and* $\mathsf{LinkG16.Link}$ *are honestly computed, then* $\mathsf{LinkG16.LinkVerify}$ *succeeds.*

Proof. We show that the $\mathsf{LinkVerify}$ equation above holds for all i. For legibility, we omit the index i in the proof. Suppose π_{link} is computed honestly, i.e., that all U' and (A', B', C') are well-formed and that the underlying Groth16 verification equations holds on the corresponding (A, B, C). First, we note that, since C' and U were computed honestly,

$$\begin{aligned}
&e(C', [\delta]_2) \cdot e(U, H) \\
&= e(C - [z]_1, [\delta]_2) \cdot e(\textstyle\sum a_j W_j + z[\delta]_1, H) && \text{Expanding} \\
&= e(C, [\delta]_2) \cdot e(-[z]_1, [\delta]_2) \cdot e(z[\delta]_1, H) \cdot e(\textstyle\sum a_j W_j, H) && \text{Bilinearity} \\
&= e(C, [\delta]_2) \cdot e(\textstyle\sum a_j W_j, H). && \text{Bilinearity}
\end{aligned}$$

Using this and the fact that $A' = A$ and $B' = B$, we see that the $\mathsf{LinkVerify}$ equation

$$e(A', B') = e([\alpha]_1, [\beta]_2) \cdot e(C', [\delta]_2) \cdot e(U + \hat{S}, H)$$

holds if and only if

$$\begin{aligned}
&e(A, B) \\
&= e([\alpha]_1, [\beta]_2) \cdot e(C', [\delta]_2) \cdot e(U + \hat{S}, H) && \text{Subst. } A', B' \\
&= e([\alpha]_1, [\beta]_2) \cdot e(C', [\delta]_2) \cdot e(U, H) \cdot e(\hat{S}, H) && \text{Bilinearity} \\
&= e([\alpha]_1, [\beta]_2) \cdot e(C, [\delta]_2) \cdot e(\textstyle\sum a_j W_j, H) \cdot e(\hat{S}, H) && \text{Above identity} \\
&= e([\alpha]_1, [\beta]_2) \cdot e(C, [\delta]_2) \cdot e(\hat{S} + \textstyle\sum a_j W_j, H) && \text{Bilinearity}
\end{aligned}$$

which is precisely the verification equation for the i-th underlying Groth16 instance. Since this equation holds by assumption, $\mathsf{LinkVerify}$ succeeds. □

Theorem 2. LinkG16 *is perfect HVZK.*

Proof. We define a simulator $\mathsf{Sim}_{\mathsf{linkg16}}$ with access to Groth16 trapdoors $\tau_1, \ldots, \tau_k$ as follows. For each i, sample $\bar{A}'_i, \bar{B}'_i, \bar{U}_i \leftarrow \mathbb{F}$ uniformly. Use the trapdoor τ_i to compute $C'_i \in \mathbb{G}_1$ as the unique group element which satisfies the i-th LinkVerify equation with respect to crs_i and public inputs[21] $\{a_j\}_{j=t}^{\ell}$. Concretely,

$$\bar{C}'_i := \frac{\bar{A}'_i\bar{B}'_i - \alpha^{(i)}\beta^{(i)} - \bar{U}_i - \sum_{j=t}^{\ell} a_j W_j^{(i)}}{\delta^{(i)}}.$$

For all i, let $\pi'_i := ([\bar{A}_i]_1, [\bar{B}_i]_2, [\bar{C}_i]_1)$ and $U_i := [\bar{U}_i]_1$. Finally, let

$$\pi_{\mathsf{eqwire}} \leftarrow \mathsf{Sim}_{\mathsf{eqwire}}(\{\mathsf{crs}_i, U_i\}_{i=1}^{k}).$$

The output of $\mathsf{Sim}_{\mathsf{linkg16}}$ is $(\pi_{\mathsf{eqwire}}, \{U_i, \pi'_i\}_{i=1}^{k})$.

This is indistinguishable from the real world protocol. In the real world: each U_i is uniformly distributed due to the blinding values z_i; A'_i, B'_i are uniformly distributed by the Groth16 proof procedure; and each C'_i is the unique group element which satisfies the i-th LinkVerify equation. Lastly, the simulated π_{eqwire} is indistinguishable from an honestly generated one due to the perfect HVZK of the EqWire protocol. □

Theorem 3. LinkG16 *is knowledge-sound.*

Proof. We define an extractor $\mathsf{E}_{\mathsf{linkg16}}$, aborting on verification error, as follows. By knowledge soundness of EqWire there exists an extractor $\mathsf{E}_{\mathsf{eqwire}}$ which extracts $\{a_j\}_{j=0}^{t-1}, \{z_i\}_{i=1}^{k}$ such that $U_i = z_i[\delta]_1^{(i)} + \sum a_j W_j^{(i)}$. For each $i = 1, \ldots, k$, $\mathsf{E}_{\mathsf{linkg16}}$ then reconstructs the underlying Groth16 proof

$$\pi_i = (A'_i, B'_i, C'_i + [z_i]_1).$$

$\mathsf{E}_{\mathsf{linkg16}}$ outputs $(\{a_j\}_{j=0}^{t-1}, \{\pi_i\}_{i=1}^{k})$. Since $\mathsf{E}_{\mathsf{linkg16}}$ did not abort, it is the case that, for each i,

$$\begin{aligned}
&e(A'_i, B'_i) \\
&= e([\alpha]_1^{(i)}, [\beta]_2^{(i)}) \cdot e(C'_i, [\delta]_2^{(i)}) \cdot e(U_i + \hat{S}_i, H) && \text{LinkVerify} \\
&= e([\alpha]_1^{(i)}, [\beta]_2^{(i)}) \cdot e(C'_i, [\delta]_2^{(i)}) \cdot e(z_i[\delta]_1^{(i)} + \textstyle\sum a_j W_j^{(i)} + \hat{S}_i, H) && \mathsf{E}_{\mathsf{eqwire}}\text{ output} \\
&= e([\alpha]_1^{(i)}, [\beta]_2^{(i)}) \cdot e(C'_i + [z_i]_1, [\delta]_2^{(i)}) \cdot e(\hat{S}_i + \textstyle\sum a_j W_j^{(i)}, H) && \text{Bilinearity}
\end{aligned}$$

which is precisely the verification equation for π_i. □

[21] How does the verifier know a_j (for $j \geq t$) if it was only given the prepared input $\hat{S}$? It is merely for brevity that LinkVerify is written to take $\hat{S}$. The verifier always knows the prepared input's constituent a_j values.

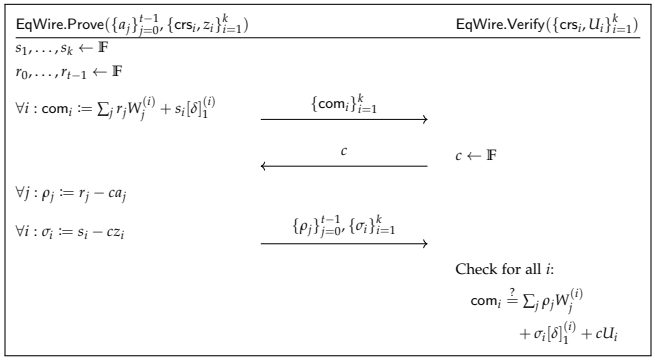

Figure 8: The EqWire protocol

F EqWire

We now define and prove secure a proof system for the discrete-logarithm equality relation,

$$R_{\mathsf{eqwire}} = \left\{ \begin{array}{c} (\{U_i, \mathsf{crs}_i\}_{i=1}^{k}; \{a_j\}_{j=0}^{t-1}, \{z_i\}_{i=1}^{k}) : \\ \bigwedge_{i=1}^{k} U_i = z_i[\delta]_1^{(i)} + \sum_{j=0}^{t-1} a_j W_j^{(i)} \end{array} \right\}.$$

The proof system is instantiated using a proof framework due to Camenisch and Stadler [CS97]. Concretely, it is the Fiat-Shamir transform of the protocol described in Figure 8.

Proofs

Theorem 4. *The* EqWire *protocol is knowledge-sound.*

Proof. We define extraction in the usual way for Camenisch-Stadtler sigma protocols. Let $\mathsf{E}_{\mathsf{eqwire}}$ be our extractor, aborting on verification failure. The extractor receives the commitments, and then picks challenge $c \leftarrow \mathbb{F}$. It sends c and receives $\{\rho_j, \sigma_i\}_{i,j}$. The extractor then rewinds to pick a fresh $c' \leftarrow \mathbb{F}$. It sends c' and receives $\{\rho'_j, \sigma'_i\}_{i,j}$. For all i and j, the extractor computes

$$a_j \coloneqq \frac{\rho_j - \rho'_j}{c' - c} \qquad\qquad z_i \coloneqq \frac{\sigma_i - \sigma'_i}{c' - c}$$

and outputs $(\{a_j\}_{j=0}^{t-1}, \{z_i\}_{i=1}^{k})$. Since the extractor did not abort, i.e., both runs passed verification, and the commitments did not change, it is the case that $U_i = z_i[\delta]_1^{(i)} + \sum_j a_j W_j^{(i)}$ for all i. □

Theorem 5. *The* EqWire *protocol is perfect HVZK.*

Proof. We define a simulator as follows: sample c and all σ_i, ρ_j uniformly from $\mathbb{F}$; for all i, compute $\mathsf{com}_i := \sum_j \rho_j W_j^{(i)} + \sigma_i[\delta]_1^{(i)} + cU_i$; output $(c, \{\mathsf{com}_i, \rho_j, \sigma_i\}_{i,j})$.

This is perfectly indistinguishable from a real transcript: all σ_i and ρ_j are uniform iid since they are blinded by s_i and r_j, respectively; c is uniform and independent by definition of honest-verifier; and all com_i are uniquely determined by these values. In the simulator, each σ_i and ρ_j is uniform iid by construction, and com_i is uniquely determined by these values. □

G Instantiation with Pedersen hashing

In Table 5 we give benchmarks using a different collision resistant hash for the Merkle tree. Specifically we use a Pedersen hash over the Jubjub curve.

Client-opt. (C)	ShowCred			VerifyShow			Proof Size	
Server-opt. (S)	C	C (full)	S	C	S	S (batch)	C	S
Simple Possession	5ms	784ms	699ms	4ms	1.5ms	1.8 verifs/ms	744B	192B
Expiry	98ms	875ms	796ms	6ms			1064B	
Linkable Show	104ms	879ms	837ms					
Rate Limiting	117ms	895ms	817ms					
Clone Resistance	139ms	916ms	812ms					

Table 5: Gadget microbenchmarks using Pedersen hashes for two versions of `zk-creds`. Membership proofs are done on a Merkle forest of size 2^{31} (tree height = 24, #trees = 2^8). The first configuration (C) minimizes client-side proving cost; the second configuration (S) maximizes server throughput. ShowCred (full) gives the cost of including membership recomputation while showing a credential. VerifyShow (batch) gives throughput for verifying a set of 100 proofs. We emphasize that all verification numbers are *single-threaded*, allowing for efficient concurrent processing.

H CRS sizes of evaluated circuits

We list in Table 6 the sizes of the proving and verifying keys for all the circuits evaluated in Sections 6 and 7.

	Proving key (MB)	Verifying key (KB)
Forest membership	0.3	12.8
Tree membership	10.1	0.6
Expiry	1.5	0.6
Linkable show	0.4	0.6
Rate limiting	1.3	0.8
Clone resistance	1.2	0.8
Age-restricted vid.	2.8	0.9
Entering a bar	2.5	0.6

Table 6: CRS sizes for our evaluated circuits