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

July 4, 2022

**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 make (often incorrect) assumptions, e.g., that the local department of motor vehicles will 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 via 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 300ms in a real-world scenario of using a passport to anonymously access age-restricted videos.

# 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, in fact, are not true in practice. The designs arising from these assumptions make anonymous credential schemes difficult to actually deploy.

Existing anonymous credential schemes make, at a minimum, some subset of the following incorrect assumptions: there is a single issuer for a given identity property; 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 issuing credentials; all use cases and attributes needed for a credential are known in advance at system setup and are only simple statements ("my age is"); 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 functionality provided by general purpose zero-knowledge proofs (specifically Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge or zkSNARKs) lets us address most of these challenges and support real-world requirements, such as allowing for arbitrary issuers with different credential forms and post-hoc composition of credentials and new access criteria.

[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 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 ACS is also being considered for private groups in Signal [CPZ20].

## 1.1 Past work and real-world limitations

Several approaches have tried to address the limitations of anonymous credentials, focusing primarily on issuance, but with little real-world success.

**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 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 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. We do not even need full anonymous credentials, simple Chaumian blind tokens [CFN90] are sufficient. But in practice, a number of problems arise.

First, there is not a single source of identity documents (e.g., the US has 50+ drivers licenses) and few will 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.

Fourth, new identity documents need to be integrated as they emerge to avoid access equity issues, and these documents may have differing for-

[3]Consider something as simple as date formats: Japanese Drivers Licenses give birth year relative to eras that, currently, correspond to the reign of the emperor. However, they use the Gregorian calendar for months and days.

mats. 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.

Real world usage of anonymous credentials requires protocols that hide more than just a signature or some attributes. We need systems that are flexible, dynamically adaptable to new use cases post-deployment, and can support complex access criteria that are agnostic to the issuer or credential format.

## 1.2 Credentials from zero-knowledge proofs

Switching to zero-knowledge proofs as the basis for anonymous credentials, instead of blind signatures, represents a paradigm shift: rather than imagining a subset of use cases and designing custom protocols for each while balancing cryptographic tradeoffs, we can get full privacy and full expressivity after the protocol is designed and deployed.

Anonymous credentials built from general purpose zero-knowledge proofs can hide the additional contextual information that arises from the complexities of real world usage: they allow protocols that hide why a user is eligible for a credential when obtaining it, hide which credential the user shows to meet some access requirement, and hide all information in the user's credential beyond that it meets some access criteria.

The development of zkSNARKS in the past ten years, both in terms of efficient implementations, and perhaps even more significantly, software support for defining proof statements, has made a zero-knowledge credential approach not just realizable, but practical. Because zkSNARKs efficiently support any (small) NP relation, we no longer need to design custom protocols with limited features to achieve acceptable performance for many applications.

## 1.3 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` is the realization we can use general-purpose but efficient zero-knowledge proofs to build deployable credential schemes and meet the above design goals.

Zero-knowledge proofs enable `zk-creds` to support flexible and composable access criteria. `zk-creds` not only allows users to privately show their

credential(s) meet some arbitrary access criteria check, these access criteria can be defined at any time (even after system setup or credential issuance), by any party, and composed dynamically via *gadgets*. This flexibility allows `zk-creds` to meet the reality of real-world identity applications: that requirements can dynamically change at any time, and all issuers for a given attribute or use cases for that attribute might not be known in advance (or ever).

The second major contribution of `zk-creds` is its support of existing government identity infrastructure without modification or collaboration. Using 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. We call this *zk-supporting-documentation* and it allows us to provide a digital analog of walking into a US Department of Motor Vehicles and presenting existing identity documents to get a driver's license, but 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 (e.g., a transparency log or blockchain) and each credential can include zk-supporting-documentation justifying its issuance, the set of all issued credentials is auditable. This is not possible when credentials can be surreptitiously issued via signing keys.[4]

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 $< 300ms$
- 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 that supports proof re-randomization and enables proof reuse
- includes a full application for age-restricted video access with cloning resistance, using existing passports for issuance.

[4]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.

### 1.4 Outline of this work

The rest of this work proceeds as follows. We provide an overview of `zk-creds` in Section 2. We introduce the necessary cryptographic preliminaries in Section 3. We present the definition of `zk-creds` and an ideal functionality defining its security in Section 4, and then we present our construction in Section 5. Section 6 describes our implementation and evaluation. In Section 7, we detail two example case studies for `zk-creds`, and we discuss a variety of potential real world applications in Section 8. Finally, we review related work in Section 9 and conclude in Section 10.

## 2 Overview

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

A *credential* in `zk-creds` is a commitment to a set of arbitrary attributes such as the fields from a passport (name, date of birth, etc.). Unlike traditional anonymous credential schemes, credentials are issued not with signatures, but by being placed on a list.

During issuance, a user convinces a credential issuer they are entitled to a credential. The *issuer* is the maintainer of the list and may be a single trusted party, a Byzantine system, or a public blockchain. The mechanism for credential issuance decisions and the credential format is application-specific. `zk-creds` supports arbitrary attribute formats and arbitrary zk-supporting-documentation for decisions. For example, a credential for Sybil resistance might be issued in response to a CAPTCHA, while our age verification credential requires a zero-knowledge proof with respect to a US passport.

Credential issuance is auditable: anyone can download the list of credentials and, with zk-supporting-documentation, verify issuance. Even without such documentation, all issued credentials are visible and can be investigated. In contrast, it is impossible to enumerate, let alone audit, every credential signed with a given key.

Our approach requires the user to maintain an up-to-date witness to the credential's membership in the issued list. The user can, periodically, ask the issuer for an updated witness. Looking ahead, because our membership list is implemented with Merkle trees, the user can also periodically download the logarithmic-sized *frontier* of the Merkle tree, along with some constant-sized ancestry data, and use that to update their witness. They need not download or store the full list.

Users, once issued a credential, *show* they meet some *access criteria* such as being over 18. They produce a zero-knowledge proof that their credential 1) is in a Merkle tree of all issued credentials, and 2) meets the access criteria. Anyone can then *verify* the proof against the credential list and access criteria

without further interaction. Importantly, application developers can define arbitrary access criteria at any point in the life of the system. We support common features from the anonymous credential literature, such as hidden attribute credentials, inequality or expiry checks, rate limiting, and clone resistance where violating the rate limit (e.g., by sharing a credential with others) results in the credential's identification and revocation. Because access criteria can be any NP relation in our scheme, we can 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.

Finally, if a credential needs to be revoked for any reason, an issuer may simply remove it from their list. This captures so-called *public key revocation* where the holder's identity is known, and private key revocation where a stolen or leaked credential is banned. In contrast, many existing schemes require work, specifically expensive public key operations, linear in the number of issued or revoked credentials.

# 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 $\mathsf{A}$ on input $x$, using randomness $r$. 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 $\lambda$.

## 3.2 Merkle trees

In `zk-creds` we use Merkle trees to represent set membership. The root of a tree $T$ is denoted $T_{\mathsf{root}}$. A Merkle *forest* $F$ is a set of roots. Merkle trees expose 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* that proves that $v \in T$. The size of $\theta$ 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 $\mathsf{bg}$. We define these terms in Appendix B and use them in the construction of $\mathsf{LinkG16}$ in Appendix C.

**Groth16.** This is a trusted-setup zkSNARK scheme introduced by Groth [Gro16]. To describe `zk-creds`, it suffices to specify the functionality of Groth16 that any general-purpose NIZK proof scheme supports:

$\mathsf{G16.Setup}(\mathsf{bg}, \mathsf{desc}) \to \mathsf{crs}$ Generates a common reference string ($\mathsf{crs}$) for the given arithmetic circuit description and bilinear group.

$\mathsf{G16.Prove}(\mathsf{crs}, x, w) \to \pi$ Proves the circuit described by $\mathsf{crs}$ is satisfied, where $x$ are the public inputs and $w$ are the witnesses.

$\mathsf{G16.Verify}(\mathsf{crs}, \pi, x) \to \{0,1\}$ Verifies the proof $\pi$ w.r.t. the given public inputs.

**Groth16 Linkage.** We describe a high-level interface that allows us to construct a *linkage* proof over Groth16 proofs. This allows one to show that a hidden collection of Groth16 proofs all share some subset of hidden public inputs.

$\mathsf{LinkG16}$ is instantiated as a Camenisch-Stadler sigma protocol [CS97], proving that all Groth16 verification equations are satisfied without revealing the Groth16 proof elements or linked public inputs themselves. See Appendix C for the full description and security proofs of $\mathsf{LinkG16}$. For `zk-creds`, however, it suffices to specify the functionality:

$\mathsf{LinkG16.Link}(x^*, \{\mathsf{crs}_i, \pi_i\}_{i=1}^k) \to \pi_{\mathsf{link}}$ Proves, in zero-knowledge, that

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

where $x^*$ and $\{\pi_i\}$ are hidden, and $\{x_i\}$ are public.

$\mathsf{LinkG16.LinkVerify}(\pi_{\mathsf{link}}, \{\mathsf{crs}_i, x_i\}_{i=1}^k) \to \{0,1\}$ 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 statistically knowledge-sound against algebraic adversaries under the $q-\mathsf{dlog}$ assumption [FKL18].

We use a Pedersen hash to instantiate Merkle trees, and Pedersen commitments for credentials. The tree root and credential commitment are binding if the hash function is collision-resistant, which holds under the $\mathsf{dlog}$ assumption in $\mathbb{G}$. The commitment is perfectly hiding.

Finally, we assume that the key-prefixed Poseidon hash function, used to instantiate the gadgets in Section 5.3, is a PRF.

# 4 Definitions

## 4.1 Security definitions

Security definitions are given by an ideal functionality in Figure 1. It corresponds to the usual security properties of anonymous credentials: unforgeability, correctness and unlinkability. Our ideal functionality 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.

## 4.2 Anonymous credentials

We give an overview of the data structures and algorithms exposed in our anonymous credential scheme.

Let a *credential* $\mathsf{cred}$ be the commitment $\mathsf{cred} := \mathsf{Com}(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs}; r)$ where: $\mathsf{nk}$ is the *pseudonym key*, a private random value used to generate persistent pseudonyms; $\mathsf{rk}$ 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.

We say credentials are *issued* by a party A if they appear on A's credential list, which we represent as a forest of Merkle trees. Every issuer has some *issuance criteria* that the requester must meet in order to have their $\mathsf{cred}$ issued, e.g., that the birth date in $\mathsf{cred}$ matches a signed digital passport. The requester (running $\mathsf{IssueReq}$) sends $\mathsf{cred}$ to the issuer with some *zk-supporting-documentation* ($\mathsf{sd}_{\mathsf{zk}}$) that convinces the issuer of the criteria, e.g., a Groth16 proof or a digital signature. The issuer runs $\mathsf{IssueGrant}$ and, upon success, adds $\mathsf{cred}$ to their list and returns the Merkle authentication path.

Next, let the list of 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 instantiation. A user *shows* a credential (running $\mathsf{ShowCred}$) by presenting a zero-knowledge proof that they have a valid credential and its attributes satisfy the verifier's access criteria. The verifier runs $\mathsf{VerifyShow}$ and, upon success, grants access to the user.

Let $T$ be an arbitrary credential list—for example, an accumulator or ledger—containing the credentials. Let $\mathsf{sd} := (\mathsf{sd}_{\mathsf{zk}}, \mathsf{sd}_{\mathsf{pub}})$ denote issuer-defined zero-knowledge and public supporting documentation which helps attest to validity of a credential's attributes.

An anonymous credentials system can then be defined by the following algorithms:

$\mathcal{F}.\mathsf{IssueSetup}_I(\cdot)$:

1. Decide on relevant issuance criteria $\iota$.
2. Let $\mathsf{IssueCriteria}[I] := \mathsf{IssueCriteria}[I] \cup \{\iota\}$
3. Publish $(I, \iota)$

$\mathcal{F}.\mathsf{IssueReq}_U(\iota, \mathsf{attrs}, w_{\mathsf{sd}}, \mathsf{iaux}_{\mathsf{sd}})$:

1. Sample random $\mathsf{cred}$, $r$ // commitment and its opening
2. $\mathsf{UserCreds}_U[\mathsf{cred}] := (\iota, r, \mathsf{attrs})$ // construct credential
3. Let $\mathsf{HiddenZKSD}_U[r] := w_{\mathsf{sd}}$
4. Send $(\mathsf{cred}, \mathsf{iaux}_{\mathsf{sd}})$ to $\mathcal{F}.\mathsf{IssueGrant}$
5. If the previous step aborts or returns $\perp$, abort
6. Else, it returns $\theta$: send $(\mathsf{cred}, r, \theta)$ to $U$

$\mathcal{F}.\mathsf{ShowCred}_U(\phi, \mathsf{cred}, \theta, r, \mathsf{aux})$:

1. Sample random $s$
2. Let $(\iota, r', \mathsf{attrs}) := \mathsf{UserCreds}_U[\mathsf{cred}]$
3. Check $r = r'$ and abort if it fails
4. If user $U$ is honest: let $\phi' := \phi$ and $\theta' := \theta$
5. Else, if $U$ is corrupted: let $\phi', \theta'$ be arbitrary
6. Let $\mathsf{ShowProofs}_U[s] := (\phi', \mathsf{cred}, \theta', r, \mathsf{aux})$
7. Send $(s, \mathsf{aux})$ to user $U$

$\mathcal{F}.\mathsf{RevokeCred}_I(\mathsf{cred})$:

1. Find index $\theta$ such that $\mathsf{IssuedCreds}[\theta] = \mathsf{cred}$; else, abort
2. Let $\mathsf{IssuedCreds}[\theta] := \mathsf{nil}$
3. Notify all parties that $\mathsf{cred}$ has been revoked

$\mathcal{F}.\mathsf{ShowSetup}_V(\cdot)$:

1. Decide on relevant access criteria $\phi$.
2. Let $\mathsf{AccessCriteria}[V] := \mathsf{AccessCriteria}[V] \cup \{\phi\}$
3. Publish $(V, \phi)$

$\mathcal{F}.\mathsf{IssueGrant}_I(\iota, \mathsf{cred}, \mathsf{iaux}_{\mathsf{sd}})$:

1. If issuer $I$ is honest and user $U$ is corrupted:
   (a) Let $(\iota', r, \mathsf{attrs}) := \mathsf{UserCreds}_U[\mathsf{cred}]$
   (b) Let $w_{\mathsf{sd}} := \mathsf{HiddenZKSD}_U[r]$
   (c) Check that $\iota = \iota'$ and $\iota \in \mathsf{IssueCriteria}[I]$
   (d) Check that $\iota(\mathsf{attrs}, \mathsf{iaux}_{\mathsf{sd}}, w_{\mathsf{sd}}) = 1$
   (e) If any check fails: send $\perp$ to $U$ and abort
2. Arbitrarily, $I$ may choose to deny $\mathsf{cred}$; if so, abort
3. Sample random $\theta$
4. Let $\mathsf{IssuedCreds}[\theta] := \mathsf{cred}$
5. Send updated $\mathsf{IssuedCreds}$ to $I$
6. Send $\theta$ to $U$
7. Notify all parties that $\mathsf{cred}$ has been issued

$\mathcal{F}.\mathsf{VerifyShow}_V(\phi, s, \mathsf{aux})$:

1. Let $(\phi', \mathsf{cred}, \theta, r, \mathsf{aux}') := \mathsf{ShowProofs}_U[s]$
2. Let $\mathsf{cred}' := \mathsf{IssuedCreds}[\theta]$
3. Let $(\cdot, r', \mathsf{attrs}) := \mathsf{UserCreds}_U[\mathsf{cred}]$
4. Check that $\phi = \phi'$ and $\phi \in \mathsf{AccessCriteria}[V]$
5. Check that $\mathsf{cred} = \mathsf{cred}' \neq \mathsf{nil}$, $r = r'$, and $\mathsf{aux} = \mathsf{aux}'$
6. If verifier $V$ is honest and user $U$ is corrupted: check $\phi(\mathsf{attrs}, \mathsf{aux}) = 1$
7. If any check fails: send false to $V$ and abort
8. Arbitrarily, $V$ may choose to deny $\mathsf{cred}$; if so, abort
9. Else, send true to $V$

Figure 1: An ideal functionality $\mathcal{F}$ for `zk-creds`.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ Generates the system parameters and empty credential list.

$\mathsf{IssueSetup}(\mathsf{pp}) \to \iota$ Run by an issuer to establish the *public* attribute fields and issuance criteria $\iota$ for obtaining a credential.

$\mathsf{ShowSetup}(\mathsf{pp}) \to \phi$ Run by a verifier to establish the access criteria $\phi \in \Phi$ for showing a credential.

$\mathsf{IssueReq}(\mathsf{pp}, T, \iota, \mathsf{attrs}, w_{\mathsf{sd}}, \mathsf{iaux}_{\mathsf{sd}}) \to (\mathsf{cred}, \mathsf{sd})$ Run by a user to create and request a credential under issuance criteria $\iota$ with supporting documentation $\mathsf{sd}$.

$\mathsf{IssueGrant}(\mathsf{pp}, T, \iota, \mathsf{cred}, \mathsf{sd}) \to (T', \theta)$ Run by an issuer to grant a user the requested credential and add it to the list.

$\mathsf{ShowCred}(\mathsf{pp}, T, \phi, \mathsf{cred}, \theta, r, \mathsf{ctx}) \to (\pi_{\mathsf{link}}, \mathsf{aux})$ Run by a user to show that an issued credential satisfies access criteria.

$\mathsf{VerifyShow}(\mathsf{pp}, T, \phi, \pi_{\mathsf{link}}, \mathsf{aux}, \mathsf{ctx}) \to b$ Run by a verifier to validate a credential show.

$\mathsf{RevokeCred}(\mathsf{pp}, T, \mathsf{cred}) \to T'$ Run by an issuer to revoke 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, Alice convinces the issuance mechanism she should be given a credential. Once her credential is put on the issued 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 Alice's credential is on the issued list, and (2) a proof that the committed attributes meet some access criteria. Finally, an issuer is also 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 2.

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

1. Choose bilinear group $\mathsf{bg}$ w/ large prime order $p$
2. Let $\mathsf{desc}_T$ 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 $\theta$ attesting to $\mathsf{cred} \in T$
3. Let $\mathsf{desc}_F$ 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 $\mathsf{pp}$

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

1. Sample $r$, $\mathsf{nk}$, $\mathsf{rk}$ // 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 $\theta$ attesting to $\mathsf{cred} \in T'$
9. Store $(\mathsf{nk}, \mathsf{rk}, r, \mathsf{attrs}, \theta)$

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

1. If necessary, populate private $w_i$ w/ credential contents:
   $(\mathsf{nk}, \mathsf{rk}, \mathsf{attrs}) := \mathsf{Open}(\mathsf{cred}; r)$
2. For all $i = 1, \ldots, k$, compute the access criteria proof:
   $\pi_i := \mathsf{G16.Prove}(\mathsf{crs}_{\phi_i}, (T_{\mathsf{root}}, \mathsf{cred}, \mathsf{aux}_i), w_i)$
3. Prove $\pi_T := \mathsf{G16.Prove}(\mathsf{crs}_T, (T_{\mathsf{root}}, \mathsf{cred}), \theta)$ // tree proof
4. Prove $\pi_F := \mathsf{G16.Prove}(\mathsf{crs}_F, (T_{\mathsf{root}}, \mathsf{cred}, F), \mathsf{nil})$ // forest
5. Let $(\pi_{k+1}, \pi_{k+2}) := (\pi_T, \pi_F)$ // collect the proofs
6. Let $(\mathsf{crs}_{k+1}, \mathsf{crs}_{k+2}) := (\mathsf{crs}_T, \mathsf{crs}_F)$ // collect the CRSs
7. Prove $\pi_{\mathsf{link}} := \mathsf{LinkG16.Link}((T_{\mathsf{root}}, \mathsf{cred}), \{\mathsf{crs}_{\phi_i}, \pi_i\}_{i=1}^{k+2})$
8. Send $(\pi_{\mathsf{link}}, \{\mathsf{aux}_i\}_{i=1}^{k+2})$ to user $U$

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

1. Decide on issuance criteria $(\iota_{\mathsf{zk}}, \iota_{\mathsf{pub}})$ for supporting docs.
2. Let $\mathsf{desc}$ be a circuit w/ public input $\mathsf{iaux}_{\mathsf{zk}}$ which asserts:
   $\iota_{\mathsf{zk}}(\mathsf{attrs}, \mathsf{iaux}_{\mathsf{zk}}) = 1 \wedge \mathsf{cred}$ 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 $\phi$ for satisfying attributes.
2. Let $\mathsf{desc}$ be a circuit w/ public input $(T_{\mathsf{root}}, \mathsf{cred}, \mathsf{aux})$ 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 $\mathsf{crs}_\phi$

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

1. Let $T' := T.\mathsf{Remove}(\mathsf{cred})$
2. Store $T'$ and update the forest $F$
3. Return $T'$

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

1. Let $(\pi_\iota, \mathsf{iaux}_{\mathsf{zk}}) := \mathsf{sd}_{\mathsf{zk}}$.
2. Check $\iota_{\mathsf{pub}}(\mathsf{sd}_{\mathsf{pub}})$ // check public supporting documentation
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 $\theta := T'.\mathsf{AuthPath}(\mathsf{cred})$ // $\theta$ attests to $\mathsf{cred} \in T'$
7. Store $T'$ and update the forest $F$
8. Send $\theta$ to user $U$

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

1. Let $(\mathsf{crs}_{k+1}, \mathsf{crs}_{k+2}) := (\mathsf{crs}_T, \mathsf{crs}_F)$ // collect the CRSs
2. Let $(\mathsf{aux}_{k+1}, \mathsf{aux}_{k+2}) := (\mathsf{nil}, F)$ // collect the 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

Figure 2: `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.

## 5.1 Merkle Forests

Our credential list's membership proof makes use of a novel technique we call Merkle forests. The membership proof attests to two parts: $\mathsf{cred} \in T$ for some Merkle tree $T$, and $T \in F$ where $F$ is the forest of Merkle trees containing issued credentials. Compared to a single 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 (looking ahead: up to 22%, or 431ms). 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 ($137\mu$s).

**Witness management.** Showing a credential in `zk-creds` requires knowing the witness (i.e., the siblings of the path from the credential to the tree root) for membership in the Merkle tree. This witness must be updated as the tree changes and credentials are added and removed. In a naïve construction, a user would stream every added credential and update their witness. On the other extreme, credential holders could periodically query a third party for updated witnesses. However, in low use deployments, care must be taken to avoid correlating requests for an updated witness (which identifies the credential) with a show that is intended to hide the credential. We present two simple expedient solutions to this problem, and leave a broader discussion to a future version of this paper.

Assume credentials are added to a fixed height tree from left to right. Observe that, given a witness to membership of any credential in a tree at time $t$, subsequent updates only affect the witness where it intersects with the path from the root to the right-most element of the tree at time $t$. We call this right-most path the *frontier*. As a result, we can update any witness at time $t$ to a witness at $t + n$ by giving a single log-sized Merkle tree frontier from the current root (at $t + n$) to the last element added at $t$. By the same property, we can also represent a batch of additions by the change it has on the existing tree's frontier. Thus, credential holders need not see every added credential, they need only see log-sized periodic batch updates.

## 5.2 Blind Groth16

The membership proof is the costliest part of ShowCred. Looking ahead, it takes ~800ms 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 Groth16 linkage proof to combine a membership proof with multiple access proofs. Blind Groth16 lets us reuse an already computed membership proof in multiple shows without breaking privacy. Further, 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) 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 C.

## 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 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 natural ones that are included by default in `zk-creds`. 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 domain, but is unlinkable to any use of the credential in other domains. For example, a single Sybil-resistant credential could be used for creating unlinkable accounts across sub-forums within a single site, such as Discord groups 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 is due to Camenisch et al. [CHK$^+$06] (Sec-

tion 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, $\mathsf{tok}_1$ will be repeated, but $\mathsf{tok}_2$ will be distinct, giving the verifier two instances of the $\mathsf{tok}_2$ equation and one unknown variable: id. 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 issued list.

**Expiry** Opens an attribute $e$ in the credential and proves that $e >$ today, proving 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 to every ShowCred. This can be done by including an empty proof that takes the nonce as the public input[5].

**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 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

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

fully support hidden issuer. 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`. A credential show can simply be used as zero-knowledge supporting documentation to issue other credentials. Moreover, because the proof in a show is arbitrary, the show used for a delegated issuance can compute arbitrary values that need to be included in the delegated credential as hidden attributes. 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. Finally, credential attributes can be selectively delegated as well.

## 5.5 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 (aka 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 multiple party computation setup *ceremonies* [BGM17, BCG$^+$15, KMSV21] that need only one honest party. 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 that ensure 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.

## 5.6 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 1.

The simulator $\mathcal{S}$ first runs Setup and gets the necessary trapdoors for 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. In our case, because Groth16 proofs are simulated, we need not directly simulate the linkage proofs.

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 malicious parties can revoke credentials, so the simulator must synchronize revocations.

Assuming that Groth16 is perfectly zero-knowledge and simulation-extractable (the latter holds in the Algebraic Group Model); linkage proofs are sound and zero-knowledge under the Random Oracle Model and dlog (see Appendix C); Pedersen hashes are collision-resistant under dlog; Pedersen commitments are computationally binding under dlog and perfectly hiding; and Poseidon is a secure PRF, 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 2 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 running Ubuntu 20.04 with kernel 5.11.0-40-generic.

**Code.** `zk-creds` consists of 7.6k lines of Rust code[6] 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

[6] All code will be open sourced by August 1.

scanner app to extract intermediate cryptographic values from the passport and dump them to a JSON file. 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 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 [Bow17], and instantiate Com as a Pedersen commitment over the Jubjub curve [ZCa19]. The collision-resistant hash function used for all Merkle trees is a Pedersen hash over Jubjub. The PRFs in our multishow tokens example are domain-separated instances of the Poseidon hash function [GKK+19], configured to be compatible with BLS12-381 and a 128-bit security level ($\alpha = 3$ and capacity $= 1$).

## 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.

Figure 3a gives a summary of `zk-creds`'s performance for common usage scenarios. We assume a setting where $2^{31}$ credentials have been issued. A Basic Show takes 5ms to produce and 4ms to verify, assuming precomputation of the Merkle tree and forest membership proofs. More complex statements like rate-limited credentials with clone resistance take 140ms to show and 6ms to verify. If the Merkle tree membership proof is not precomputed, then the full show takes an additional 800ms 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 a server-optimal version of `zk-creds` in Figure 3b, which is optimized for verification latency and throughput. The server-optimized construction combines the Merkle membership and criteria check circuits into a single proof. 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 three times as many proofs.

| Client-opt. | ShowCred | ShowCred (full) | VerifyShow | Proof Size |
|---|---|---|---|---|
| **Simple Possession** | 5ms | 784ms | 4ms | 744B |
| **Expiry** | 98ms | 875ms | 6ms | 1064B |
| **Linkable Show** | 104ms | 879ms | | |
| **Rate Limiting** | 117ms | 895ms | | |
| **Clone Resistance** | 139ms | 916ms | | |

(a) Benchmarks for `zk-creds` configured for minimizing client side proving cost. The Merkle-forest proof is separate and can be reused across Shows. ShowCred (full) gives the cost of showing a credential inclusive of membership recomputation.

| Server-opt. | ShowCred | VerifyShow | VerifyShow (batch) | Proof Size |
|---|---|---|---|---|
| **Simple Possession** | 699ms | 1.5ms | 1.8 verifs/ms | 192B |
| **Expiry** | 796ms | | | |
| **Linkable Show** | 837ms | | | |
| **Rate Limiting** | 817ms | | | |
| **Clone Resistance** | 812ms | | | |

(b) Benchmarks for `zk-creds` configured for maximizing server throughput by using a single monolithic SNARK without proof reuse. VerifyShow (batch) gives throughput for verifying a set of 100 proofs. We emphasize that all verification numbers are *single-threaded*, allowing efficient concurrent processing.

Figure 3: Gadget microbenchmarks. Membership proofs are done on an issuance list of size $2^{31}$ (tree height = 24, #trees = $2^8$).

# 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, ICA21]. 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 $\mathsf{DG}_1$ and $\mathsf{DG}_2$.

Parsing the contents of an e-passport in zero-knowledge is non-trivial: the signature is not just over $\mathsf{DG}_1$ and $\mathsf{DG}_2$, 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 $\mathsf{DG}_3$ through $\mathsf{DG}_{16}$, 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 $\mathsf{DG}_2$.[7] Since $\mathsf{DG}_1$ contains the attributes we care about, we must parse $\mathsf{DG}_1$ inside the zero-knowledge proof. Luckily, since $\mathsf{DG}_1$ 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.[8] Computing this proof takes less than 3 seconds.

**Why zk-proofs over passports are insufficient.** We cannot just use the zero-knowledge passport proof as a credential for accessing age restricted content: to prevent credential sharing, we need cloning resistance, which requires credentials include a secret random seed that a passport lacks.

## 7.2 Instantiating the bulletin board

To instantiate `zk-creds`, we only need a bulletin board and parties running our software. We instantiate our bulletin board using a smart contract on an

[7] In reality, $\mathsf{DG}_2$ contains slightly more than the bare image; it also has the image's creation and expiry dates.

[8] We heuristically assume the hash is sufficiently random, given that the passport photo is secret, to hide its content. We note that revealing the econtent hash reveals, at least to the passport authority, the identity of the requester. This is not a problem directly. Issuance in `zk-creds` need not be anonymous since ShowCred proofs are unlinkable to issuance. It is possible to hide the signature and hash. [OWWB20] give a circuit for RSA signature verification inside a Groth16 proof. From this we estimate that extending our issuance proof to hide the signature would add 47s to IssueReq proving time.

Ethereum Virtual Machine (EVM) compatible blockchain.

Credentials are issued by an issuer who posts the issuance request to a smart contract implementing a basic bulletin board which stores the credentials and the root of the credential Merkle tree. Auditors can download the full list of credentials and issuance requests, but, importantly, ordinary users need not do so. Periodically, users get the latest tree root from the bulletin board and ask the issuer out-of-band for the authentication path of their credential. To perform VerifyShow, a verifier need only retrieve the current Merkle tree root from the bulletin board.

Finally, we need to prevent DoS attacks that flood the bulletin board. Our current prototype assumes a smart contract operator who is authorized to add to the bulletin board contract. We could, instead, verify zk-supporting-documentation and Merkle tree root computations in the EVM smart contract. While this is feasible,[9] verifying proofs and Merkle tree updates without extensive optimizations is expensive. A second option for operator-free setup is to support on chain proof verification, but with an optimistic rollup [Eth]: bulletin board additions include a deposit which is burned if, when the smart contract evaluates a challenge, it determines the proofs or computed Merkle tree root is false. While the challenge still requires costly computation, this is not paid for in issuance.

We implement our smart contract in Solidity and the requisite client side scripting for posting and retrieving data with `web3.js`. We can deploy our contract on Ethereum or other EVM compatible chain such as Avalanche's C-Chain. Contract operations in EVM are measured in gas which can, closely, be thought of as a complexity-weighted count of EVM instructions used. Posting a full credential to the list along with issuance requests (to support auditing) costs 576,808 gas whereas posting only the credential costs 78,355 gas. Posting the Merkle root (done via a separate call) also 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 ranged from about 20 to 50 Gwei (a Giga-wei is $10^{-9}$ ETH), and 1 ETH is about $1800 USD. Posting a full issuance request and credential thus costs about $20 - $50 USD, and posting a lone credential costs about $3 - $7 USD. Effectively, 1 KB of stored credentials equates to $29-89. For Avalanche, gas prices are similar, but at $25 USD per Avalanche 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

[9]Some deployed systems, e.g., Tornado Cash [Tor], already do this for Groth16 proofs, but some engineering would be needed to adapt `zk-creds` and the underlying Arkworks Groth16 implementation to elliptic curves supported by Ethereum and port the verification logic over to be compatible with the Arkworks Groth16 implementation instead of Bellman derived schemes.

| | IssueReq | IssueGrant | ShowCred | ShowCred (full) | VerifyShow |
|---|---|---|---|---|---|
| **Age-restricted vid.** | 2.36s | 2ms | 258ms | 1.05s | 8ms |
| **Entering a bar** | 2.36s | 2ms | 228ms | 1.01s | 6ms |

Table 1: `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 Figure 3a.

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 that converts a passport into an anonymous credential. A modified open-source Android app extracts the NFC passport data and a separate program coverts 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, the credential, and zk-supporting-documentation proof. IssueGrant verifies the proof and the 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,[10] practical usage demands cloning protections. And cloning resistance requires a randomly sampled key (the *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 indeed any trusted party to add such information.

[10]When augmented to hide the passport signature.

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[11] and construct the access criteria they need using gadgets. For this scenario, the only issuer is our passport-based issuer, and the access criteria being proved are age, expiry, and non-cloning. Concretely, the access predicate is:

$$\begin{aligned}&\mathsf{dob} \leq \mathsf{today} - 18\mathrm{yrs} \ \wedge \ \mathsf{expiry} > \mathsf{today} \\ &\wedge \ \mathsf{CloneResistance}(\rho, \mathsf{nonce}, \mathsf{tok}, \ldots)\end{aligned}$$

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

Table 1 gives performance numbers. Given a credential, it takes Alice less than 300ms to show a website she is over 18 (and less than a second even when she must recompute her membership proof). The server can verify her assertion in less than 10ms. 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 Figure 3b, proving times would increase to approximately 1 second.

**Scenario 2: A cryptographer walks into a bar.** To purchase alcohol in the United States, one needs to show photo ID and proof that 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 design 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 and along with if 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:

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

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

Table 1 gives concrete costs for local computations. To both show and verify that a patron is over 21 takes less than 300ms in the common case and less than 1.1 seconds if membership proofs must be recomputed.

# 8 Extensions and applications

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

**Signature-issued credentials.** So far, we have discussed `zk-cred` in terms of credentials issued via a list: one proof gadget proves, in zero-knowledge, the credential is on the list, another gadget(s) performs some access criteria check w.r.t. the credential. This avoids the need for finding an issuer trusted to hold keys. However, the gadget-based approach is flexible, and we can replace the membership-check gadget with one that 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. We implemented a signature gadget for checking Schnorr (and therefore FROST) signatures and measured the proving time to be 129ms.

**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[12]. 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 interesting example of this comes in the form of online petitions and discussions. New York State has an online portal for discussion and petition

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

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, online resources such as ebooks from, e.g., the New York Public Library, are limited to city residents and 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 performing a point-in-polygon check[13]. 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 feasible: even if the Groth16 proof for the gadget is expensive, the resident or an outsourced prover need only compute it once. After the first time, the proof can be reused arbitrarily in future Shows.

**Sybil-resistant IDs from email or money.** Sybil accounts on a service are accounts which are all made by the same (possibly malicious) user. The primary way internet services prevent Sybil account creation is by requiring the user to consume a (hopefully) scarce resource. Common examples of scarce resources include money (making a micropayment), attention (completing a CAPTCHA), and identity (proving possession of a phone number, government ID, or email address).

`zk-creds` provides several avenues for Sybil-resistant credentials. 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.[14] 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

[13]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].

[14]As implemented, this requires strongly unforgeable signatures. RSA PKCS#1 v1.5 signatures, which appear in the passport benchmark, are strongly unforgeable under the RSA assumption [JKM18]. In cases where the signature is only existentially unforgeable, it may be possible to reveal the document hash itself if the contents are sufficiently high-entropy to avoid inversion.

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.

**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] rely 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 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 server 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 increasing infeasible as the value of the credential goes up. But, for example, as a simple Sybil prevention or anti-spam mechanism, it may be viable.

**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.

## 8.1 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, BCC+16, 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 (cf. [GGM14]), polynomial commitments [KZG10], Verkle trees [Kus18], or perhaps special purpose schemes for SNARKs [ZBK+22]. Again, for many such accumulators, this is a simple black box swap of the membership SNARK. Instantiating the `zk-creds` approach with alternative accumulators will offer different trade-offs for witness size and data update requirements, witness computation cost, as well as verification time for the accumulator (which manifests as increased proving time) and possibly for the zkSNARK itself. A particularly exciting prospect is the co-design of accumulator schemes and SNARKs to achieve drastically improved performance.

**Co-design or co-selection of zero-knowledge proofs and proving systems.** Similarly, one could instantiate either our existing version of `zk-creds`, or a different construction, with different cryptographic primitives. For example, using Poseidon as the collision resistant hash function instead of a Pedersen hash for the Merkle tree, or using some newer circuit optimized hash function, perhaps making use of low degree gates in proving systems like Plonk [GWC19].

# 9 Related work

Anonymous credentials derive from a long line of work, starting with Chaum [Cha85] and then 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 and selective disclosure of attributes. Because it uses general purpose zero-knowledge proofs, `zk-creds` can (and does) capture all of these properties, and we have implemented them.

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 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 store only logarithmic 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 does nothing to address the scarcity of possible issuers. In many settings, finding even a single party who both 1) is empowered to make identity statements and 2) is willing and able to run cryptographic infrastructure, is a challenge. 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. We note, however, that `zk-creds`, as we discuss in Section 8, can be extended to use credentials issued via threshold signatures (including both ECDSA and Schnorr).

**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 selected 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.

# 10 Conclusion

We design, build, and benchmark `zk-creds`, a flexible, issuer-agnostic anonymous credential toolkit. `zk-creds` supports complex privacy preserving identity statements and composition of credentials, without giving up the properties afforded by previous anonymous credential systems. We are able to achieve this by utilizing general purpose zero-knowledge proofs as the basis for our scheme, which represents a paradigm shift from designing custom protocols with a subset of features to allowing for full privacy and full expressivity at any point in the lifetime of the system.

We demonstrated `zk-creds` can handle real world usage through two case studies using existing government identity infrastructure: given only a mechanism for maintaining a shared list of issued credentials, `zk-creds` lets a user convert her existing passport into an anonymous credential for her birthdate, keep the passport secret, and later use the issued credential to access an age restricted website or show her face at a bar. In the website case, the site learns only that a visitor has some document from a trusted source which says the visitor is over 18, but not the visitor's birthdate, any of the other information on the ID, or if she is a returning viewer. All of these details are hidden through the use of zkSNARKS, an efficient general purpose zero-knowledge proof. Finally, to demonstrate `zk-creds` flexibility, we discuss a number of extensions and additional applications that can be captured.

# 11 Acknowledgements

We would like to thank Mary Maller for the idea and sketch of the LinkG16 proof system, and Daniel Benarroch Guenun for starting the discussion of Merkle Forests.

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 grant CNS-1816422. Ian Miers's work was supported in part by a Facebook Research Award.

# 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.

[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 *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[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.

[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.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Fischlin and Coron [FC16], pages 327–357.

[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 Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 457–475, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[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.

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

[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.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88*, pages 319–327, New York, NY, 1990. Springer New York.

[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*, 28(10):1030–1044, Oct 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 Ligatti et al. [LOKV20], pages 1445–1459.

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

[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.

[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.

[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/.

[FC16] Marc Fischlin and Jean-Sébastien Coron, editors. *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*. Springer, Heidelberg, May 2016.

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

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

[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.

[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 Fischlin and Coron [FC16], pages 305–326.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical non-interactive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[ICA21] ICAO. Machine Readable Travel Documents. Part 10: Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC). Doc. 9303, International Civil Aviation Organization (ICAO), 8th ed., 2021. https://www.icao.int/publications/Documents/9303_p10_cons_en.pdf.

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

[JKM18] Tibor Jager, Saqib A. Kakvi, and Alexander May. On the security of the PKCS#1 v1.5 signature scheme. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1195–1208. ACM Press, October 2018.

[KG21] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography*, pages 34–65, Cham, 2021. Springer International Publishing.

[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.

[LOKV20] Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors. *ACM CCS 2020*. ACM Press, November 2020.

[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.

[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.

[OWWB20] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2075–2092. USENIX Association, August 2020.

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

[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/`.

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

[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`.

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

[ZMM+20] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Ligatti et al. [LOKV20], pages 1919–1938.

| #leaves | $2^{15}$ | $2^{31}$ | $2^{47}$ | $2^{63}$ |
|---|---|---|---|---|
| **Issuance proof time** | 0.25s | 0.78s | 1.23s | 1.56s |

Table 2: Cost of a proof of credential list membership as the size of the list varies.

# 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 attribute(s) are part of the credential, 3) the attribute(s) 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).

**Commitment-opening benchmarks.** Figure 4 plots the performance of a trivial access proof, i.e., a Groth16 proof of possession of the credential opening, as the size of the committed attributes varies. This is the baseline cost of any access criteria proof in `zk-creds`, since all such proofs must include a proof of possession of the opening.

**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 membership proving time. This results in a 22% (431ms) 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 2, 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 chosen optimally as described above.

**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 Deferred preliminaries

In order to describe the LinkG16 proof system, we require preliminaries on bilinear pairings and the Groth16 proof system.

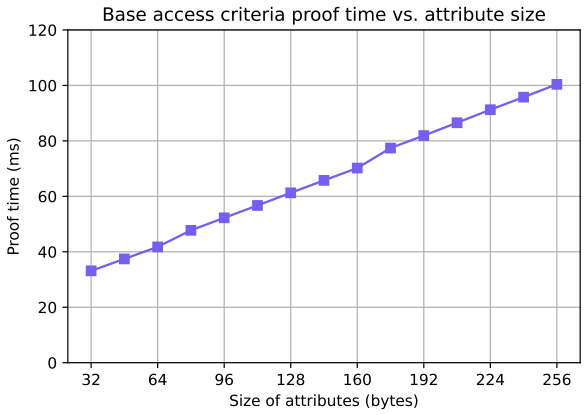


Figure 4: Cost of possession proof as the size of attrs varies.

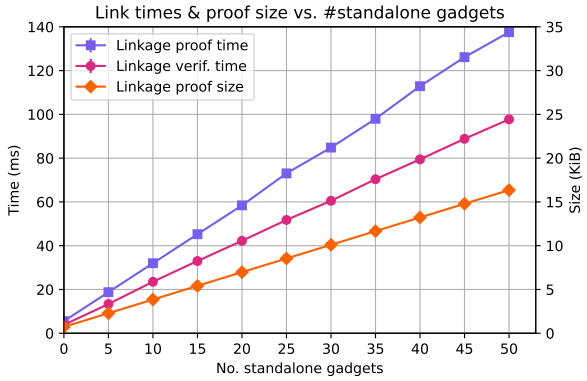


Figure 5: Costs of linkage proofs as #standalone gadgets varies.

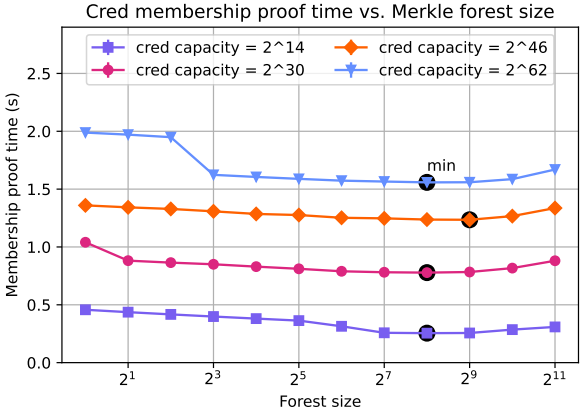


Figure 6: Costs of proving tree then forest membership, as number of trees in forest varies.

## B.1 Field and pairing notation

We will work exclusively with prime-order groups and their associated scalar fields. Group elements are denoted with capital letters $H \in \mathbb{G}$. We say $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ is a *non-degenerate type-3 bilinear group* if $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are of prime order; $G \in \mathbb{G}_1$ and $H \in \mathbb{G}_2$ are generators; and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ has the following properties:

**Bilinearity** For all $a, b \in \mathbb{Z}_p$ and $G \in \mathbb{G}_1$, $H \in \mathbb{G}_2$, $e(aG, bH) = e(G, H)^{ab}$

**Non-degeneracy** $e \neq 1$

**Type-3** There is no efficient group homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$

For a scalar $a \in \mathbb{F}$, we use $[a]_1$ and $[a]_2$ to denote $aG$ and $aH$, respectively. We use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$, and multiplicative notation for $\mathbb{G}_T$.

## B.2 Definitions

For the security proofs in the following appendices, we require definitions for soundness and zero-knowledge in proof systems. The below definitions are from [BMM+19]. $\langle \mathsf{A}, \mathsf{B} \rangle$ denotes the transcript of a protocol run between algorithms $\mathsf{A}$ and $\mathsf{B}$.

**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* $\mathsf{P}^*$*, there exists an efficient extractor* $\mathsf{E}$ *such that for all PPT adversaries* $\mathsf{A}$*,*

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

*We say* $\Pi$ *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* $(x, w, \rho)$*; and* $\mathsf{A}_1(\mathsf{tr})$ *decides whether a transcript is the result of a simulation or not.*

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

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

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

## B.3 Groth16

We describe a trusted-setup zkSNARK scheme, due to Groth[15] [Gro16], which operates over a non-degenerate type-3 bilinear group. We use $\mathbb{F}$ to denote the

[15]To support linkage, we diverge slightly from Groth's original construction by setting one of the trapdoor values $\gamma$ 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 $\gamma := 1$.

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 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. $\pi$ 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

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

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.

$\mathsf{G16.Rerand}(\mathsf{crs}, \pi) \to \pi'$ 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.

# C LinkG16

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

## C.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:

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

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

## C.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, $U_i \coloneqq z_i[\delta]_1^{(i)} + \sum_{j=0}^{t-1} a_j W_j^{(i)}$. Let $\pi_{\mathsf{eqwire}}$ be an EqWire discrete-log equality proof (described in Appendix D) that the $U_i$ commit to the same $a_j$ values,

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

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

$$\pi_i' \coloneqq (A_i, B_i, C_i - z_i G).$$

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 $\pi_{\mathsf{eqwire}}$ using EqWire.Verify. Then unpack each $\pi_i'$ into $(A_i', B_i', C_i')$. For each $i = 1, \ldots, k$, check

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

## C.3 Proofs

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

*Proof.* We show that the LinkVerify equation above holds for all $i$. For legibility, we omit the index $i$ in the proof. Suppose $\pi_{\mathsf{link}}$ 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,

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

Using this and the fact that $A' = A$ and $B' = B$, we see that the 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, 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 $\tau_i$ to compute $C'_i \in \mathbb{G}_1$ as the unique group element which satisfies the $i$-th LinkVerify equation with respect to $\mathsf{crs}_i$ and public inputs[16] $\{a_j\}_{j=t}^{\ell}$. 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 $\pi_{\mathsf{eqwire}}$ 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

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

[16] 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.

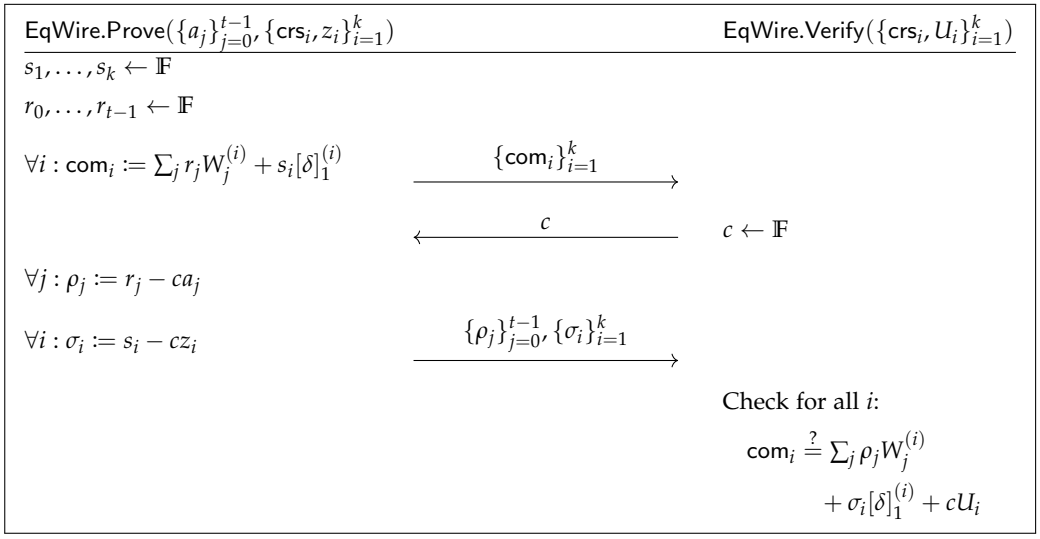


Figure 7: The EqWire protocol

$\mathsf{E}_{\mathsf{linkg16}}$ 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 $\pi_i$. □

# D EqWire

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

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

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 7.

## 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 $\sigma_i, \rho_j$ uniformly from $\mathbb{F}$; for all $i$, compute $\mathsf{com}_i \coloneqq \sum_j \rho_j W_j^{(i)} + \sigma_i[\delta]_1^{(i)} + cU_i$; output $(c, \{\mathsf{com}_i, \rho_j, \sigma_i\}_{i,j})$.

This is perfectly indistinguishable from a real transcript: all $\sigma_i$ and $\rho_j$ 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 $\mathsf{com}_i$ are uniquely determined by these values. In the simulator, each $\sigma_i$ and $\rho_j$ is uniform iid by construction, and $\mathsf{com}_i$ is uniquely determined by these values. □