On the Distributed Issuance of Credentials

Decentralizing Trust Without Sacrificing Privacy: Benchmarking Anonymous Credential Issuance

Master Thesis

Carmen González Luque
Christian Cachin
Mariarosaria Barbaraci

Universität Bern
Universitat Politècnica de Catalunya
January, 2026

Abstract

Online authentication often reveals more personal information than necessary and enables user profiling. Anonymous credentials address this by letting users prove statements about themselves through techniques like selective disclosure with unlinkability. Yet many credential systems still rely on a single trusted issuer holding a long-term signing key, creating a single point of failure, amplifying the impact of compromise and concentrating governance power. Decentralized issuance aims to spread this trust across multiple parties or public infrastructure, but doing so introduces architectural choices with practical costs and trade-offs that are hard to judge from qualitative descriptions alone.

This thesis studies how decentralized anonymous credential are issued with a focus on distributed threshold issuance. We first provide a structured comparison of representative approaches and their properties, highlighting how trust is distributed and which features are supported in practice: selective disclosure, unlinkability, revocation and Sybil-resistance/uniqueness. We then complement this analysis with a reproducible benchmark of two contrasting implementations drawn from the literature: Coconut, a threshold blind-signature scheme, and zk-creds, a SNARK-based system with circuit-defined presentations. The benchmark measures per-credential costs for issuance (credential materialization), show/prove and verify, as well as size estimates for stored credentials and verifier payload, under a controlled execution environment with warm-up and repeated runs.

Beyond reporting numbers, the evaluation characterizes how decentralization strategies shift computation between issuance and presentation and how proof and metadata artifacts shape verifier-side payload. To ensure reproducibility, experiments are automated through a dedicated experimental setup that records code revisions, environment metadata and raw measurements, then produces aggregated datasets and plots. Overall, the thesis offers both a conceptual map of the design space and empirical evidence about implementation-level cost, supporting more informed engineering decisions for privacy-preserving credential systems under realistic deployment constraints.

Contents

1 Introduction

Have you ever wondered how much information you reveal about yourself each time you authenticate online? Everyday actions such as proving you are old enough to access a service, confirming you are a member of an organization or showing eligibility for a benefit often require sharing far more personal data than strictly necessary. Even when a service only needs a simple yes/no answer, today's identity mechanisms frequently rely on persistent identifiers and repeated disclosure, making it easy to build detailed profiles of users over time.

Anonymous credentials were introduced to break this pattern. They are a well-established cryptographic primitive for privacy-preserving authentication and authorization, enabling users to prove identity-related statements (e.g., "I am over 18", "I hold a valid membership", "I am eligible to vote") without disclosing the underlying identity data and without allowing verifiers to link multiple presentations of the same credential. In other words, they shift digital identity from revealing "who you are" to proving only "what is required".

In practice, however, deploying anonymous credentials at scale often introduces a different kind of risk: many schemes rely on a trusted issuer holding a long-term secret signing key. This issuer becomes a single point of failure, compromise and governance control. If the issuer is compromised, credential forgery becomes possible. If the issuer is coercible or misbehaving, it can undermine the very guarantees that make privacy-friendly credentials attractive in the first place.

At the same time, modern identity systems increasingly demand decentralization for resilience, transparency and multi-stakeholder governance, often relying on committees, distributed ledgers or public append-only logs. This creates a natural tension: anonymity pushes toward minimizing information and linkability, while decentralization spreads trust and often introduces shared or public state.

1.1 Contribution

This thesis explores how to issue privacy-preserving credentials in settings where trust is not concentrated in a single party. We focus on distributed threshold issuance, where multiple authorities jointly issue a credential such that no single entity, or even a small enough minority, can unilaterally create or forge it, while holders can later prove statements about their attributes without revealing more information than necessary.

The central question guiding this work is:

> *When anonymous credential issuance is decentralized, what security and privacy properties are gained or lost and how do decentralization choices redistribute cost and complexity across the credential lifecycle?*

To answer this question, the thesis combines qualitative and empirical methods. First, it compares representative approaches from the state of the art and makes explicit their trust assumptions, decentralization model and supported features (e.g., selective disclosure, unlinkability, revocation and uniqueness/Sybil-resistance). Then, it evaluates practical implications through a controlled benchmark, focusing on issuance cost, presentation (show/prove) cost, verification cost and credential/proof size and discusses the resulting trade-offs in terms of deployability and operational complexity.

In this thesis the content is organized as follows.

- *Chapter 2* introduces the necessary background on anonymous credentials, threshold issuance and the system properties used throughout the comparison.
- *Chapter 3* surveys and systematizes the relevant literature, positioning the main designs considered in this work.
- *Chapter 4* describes the benchmark methodology, experimental setup and measurement procedure in detail.
- *Chapter 5* presents the results and analyzes the observed cost patterns and trade-offs.
- *Chapter 6* documents an exploratory line of work that investigates how Sybil resistance could be introduced into Coconut while preserving its core privacy goals.
- *Chapter 7* summarizes the main findings, discusses limitations and outlines open questions for future research.

2 Background

This background sets the foundations for the rest of the thesis. It begins by introducing the anonymous credential setting, the three main roles and a minimal lifecycle. Along the way, it establishes a lightweight notation for credentials, presentations and verification, so that later sections can describe different designs precisely without getting lost in implementation details.

It then reviews the cryptographic building blocks that repeatedly appear when issuance is decentralized. Zero-knowledge proofs explain how a holder can convince a verifier while keeping sensitive information hidden. Threshold cryptography explains how issuance authority can be shared across multiple authorities, so that no single party can issue credentials on its own.

It closes with a common terminology for the system properties used throughout the comparison. A common vocabulary also helps avoid confusion when different papers use similar terms in slightly different ways.

2.1 Anonymous Credentials

2.1.1 What is an Anonymous Credential?

Anonymous credentials support authentication and authorization without forcing users to reveal their full identity in every interaction. Instead of repeatedly disclosing an identifier such as a name, an account ID or a document number, a holder proves only what a verifier needs to make a policy decision. In many cases, this means proving that an attribute satisfies a condition, while keeping the rest of the attributes hidden. A core objective is that repeated presentations should not enable verifiers, even if they collude with issuers, to identify and track users across interactions [1–3]. In other words, using an anonymous credential should feel like answering a single question, not handing over a file about yourself.

2.1.2 Roles

Anonymous credential systems typically involve three roles:

- **Issuer:** checks a real world condition or eligibility rule and issues a credential that encodes a set of attributes.

- **Holder (user):** stores credentials as reusable evidence of eligibility or status and later generates presentations that reveal only what is necessary to satisfy a verifier's policy.
- **Verifier:** checks a presentation and decides whether to accept or reject the holder's request, based on the policy it enforces.

A minimal formal view. To keep later comparisons precise, we introduce a lightweight notation that captures what most systems have in common.

Let a credential encode an attribute vector

$$\mathbf{a} = (a_1, \ldots, a_\ell),$$

and let the holder have a secret x, sometimes called a link secret. Issuance produces an object cred tied to $\mathbf{a}$ and often also to x, so that possession of the credential alone is insufficient to use it without knowledge of the holder's secret.

To make later comparisons uniform and independent of implementation details, we adopt the following generic abstraction.

A useful generic model is:

$$\mathsf{cred} = (\mathbf{a}, \sigma),$$

where σ is issuer authentication material that convinces a verifier that the attributes were endorsed by an authorized issuer. In signature based systems, σ can be viewed as a signature on a commitment to $\mathbf{a}$ and the holder secret x, although concrete constructions vary across schemes [1, 2].

Abstractly, if the issuer holds a secret key sk_I with public key pk_I, issuance can be written as:

$$\mathsf{cred} \leftarrow \mathsf{Issue}(\mathsf{sk}_I, \mathbf{a}, x).$$

In decentralized settings, sk_I is not held by a single party. It is shared across multiple authorities so that only an authorized subset can jointly produce valid issuer authentication material [2].

2.1.3 Lifecycle

A minimal lifecycle has three steps:

- **Issuance:** the holder obtains a credential from the issuer. Many systems bind the credential to a holder secret and/ or key to reduce theft and replay.
- **Presentation:** the holder creates a presentation, often a proof, that demonstrates possession of a valid credential and that a required statement about its attributes holds.
- **Verification:** the verifier checks the presentation and accepts or rejects the holder's request according to its policy. Depending on the system, verification may also include revocation checks.

A minimal formal view. Different papers use different notation, so here we define a generic interface for presentations that will be reused throughout the thesis.

The verifier specifies a policy it wants to enforce. The policy can be expressed as a predicate $\varphi(\mathbf{a})$. In addition, the verifier may require that certain attributes be revealed, while allowing the rest to remain hidden. We capture this with a disclosure set $D \subseteq \{1, \ldots, \ell\}$, which indicates which attribute positions are revealed.

Given a credential cred, a holder secret x and the policy description, the holder produces a presentation:

$$(\mathbf{a}_D, \pi) \leftarrow \mathsf{Prove}(\mathsf{cred}, x, D, \varphi),$$

where $\mathbf{a}_D$ are the disclosed attributes and π convinces the verifier that the holder possesses a valid credential consistent with the disclosed values and that the predicate $\varphi(\mathbf{a})$ holds, without revealing the remaining attributes.

The verifier checks the presentation and outputs accept or reject:

$$\mathsf{Verify}(\mathsf{pk}_I, \mathbf{a}_D, D, \varphi, \pi) \in \{0, 1\}.$$

2.2 Cryptographic Building Blocks for Decentralized Issuance

This section introduces the cryptographic building blocks that appear repeatedly when anonymous credentials are issued without relying on a single authority. The goal is not to present full protocols, but to clarify the basic interfaces and assumptions that later sections build on.

2.2.1 Selective Disclosure

Selective disclosure is the ability for a credential holder to reveal only a subset of credential attributes, or to prove predicates over them, without disclosing the remaining information. It is a core requirement of anonymous credential systems, as it allows verifiers to learn exactly what is necessary to enforce a policy and nothing more.

In practice, selective disclosure is typically implemented using zero-knowledge proofs. Rather than sending the full credential to the verifier, the holder produces a presentation that reveals selected attributes and proves that the hidden attributes satisfy a required predicate. Common examples include range proofs (e.g., age $\geqslant 18$), set membership proofs or equality proofs across credentials.

Different credential systems support different forms and levels of selective disclosure. Some systems allow revealing individual attributes, while others support expressive predicate logic over hidden attributes. These design choices later influence usability, proof size and computational cost, and therefore play an important role in system comparison and deployment decisions.

2.2.2 Zero-knowledge Proofs

A zero-knowledge proof is a protocol that lets a prover convince a verifier that a statement is true without revealing the secret information that makes it true. In anonymous credentials, this is what makes selective presentation possible. Instead of sending the full credential, the holder sends only the attributes that must be disclosed and a proof that the credential is valid and that the verifier's policy holds [3].

Using the notation introduced earlier, the presentation proof π captures the idea that there exist hidden values consistent with the disclosed ones and with the issuer endorsement:

$$\exists\, (\mathsf{cred}, x, \mathbf{a}_{\overline{D}}) \;:\; \mathsf{IssuerValid}(\mathsf{pk}_I, \mathsf{cred}) = 1 \;\wedge\; \varphi(\mathbf{a}) = 1,$$

while revealing only $\mathbf{a}_D$. Different systems mainly differ in how issuer validity is represented and checked and in how expressive the policy language is.

From a system perspective, zero-knowledge proofs act as the mechanism that enables selective disclosure, they allow a holder to selectively reveal attributes or prove predicates over them while keeping the remaining credential information hidden.

Abstractly, zero-knowledge proofs provide the presentation interface of anonymous credential systems: they define how a holder demonstrates credential validity and policy compliance to a verifier without revealing unnecessary information.

2.2.3 Threshold Cryptography for Credential issuance

Threshold cryptography distributes a secret capability across n authorities so that any subset of size at least t can act, while smaller subsets cannot. In the context of credential issuance, the shared capability is the power to produce issuer authentication material, typically a signature or signature like object, that makes a credential valid [2].

A convenient abstract view is that each authority i produces a partial contribution σ_i. A holder or an issuing coordinator collects contributions from a subset $S \subseteq \{1, \ldots, n\}$ and combines them into a single object σ whenever $|S| \geq t$:

$$\sigma = \mathsf{Combine}\big(\{\sigma_i\}_{i \in S}\big).$$

Under the threshold assumption, any coalition with fewer than t authorities cannot produce a valid σ [2].

In practice, threshold issuance decomposes into three conceptual steps. First, the issuer signing key is represented using secret sharing, so that each authority holds only a share of the key rather than the full secret. Second, distributed key generation is used to create these shares jointly, ensuring that no single party ever learns the complete signing key. Finally, threshold signing defines how authorities use their key shares to produce partial signatures, which can then be combined into a single valid issuer signature without reconstructing the secret key.

From a system perspective, threshold cryptography replaces the assumption of a single trusted issuer with an honest-threshold assumption, at the cost of additional coordination and availability requirements during issuance.

2.3 System Properties and Terminology

To compare decentralized anonymous credential approaches consistently, we use the following terms throughout the thesis:

- **Issuance model:** how a credential is created and delivered to the holder, including who participates, whether issuance is interactive and whether the issuer learns the final credential and/or hidden attributes during issuance.
- **Decentralization and governance:** what power or responsibility is distributed, how participants are selected and which assumptions are required for correctness and security. Governance also includes how the system handles upgrades, parameter changes and dispute resolution.
- **Sybil-resistance:** the ability of a system to prevent a single entity from obtaining multiple credentials or identities in order to gain unfair advantage.
- **Unlinkability:** the inability of a verifier, or colluding verifiers, to determine whether two presentations originate from the same holder and/or credential, unless linkability is explicitly allowed by the system design.
- **Selective disclosure:** the ability to reveal only selected attributes or to prove predicates over attributes (e.g., age ≥ 18) without disclosing the underlying values.
- **Revocation:** the ability to invalidate credentials after issuance and the associated cost of checking validity over time.
- **Accountability:** the extent to which misuse or policy violations can be detected or addressed under the system's security and trust assumptions.
- **Legacy identity compatibility:** how easily the approach integrates with existing identity and authentication ecosystems and whether adoption requires major changes to organizational processes.

- **Public auditability:** the extent to which system state and actions can be publicly verified to improve transparency, and the privacy trade-offs this may introduce.

3
Existing Systems

This chapter reviews existing research on decentralized anonymous credential systems, aiming to contextualize our work within the broader landscape of privacy-preserving identity management. It examines how different approaches have attempted to reconcile user privacy with authentication and authorization, identifying the main design goals, cryptographic constructions and trade-offs among them.

These systems enable users to prove identity-related statements without revealing unnecessary personal data. Building on this foundation, our work focuses on decentralized variants that distribute trust among multiple authorities or public infrastructures (e.g., blockchains), thereby enhancing privacy, transparency and resilience while eliminating single points of failure.

To illustrate the evolution of this field, we analyze four representative systems, each exploring distinct design assumptions and cryptographic mechanisms:

- **Coconut** [2]: a threshold issuance scheme based on blind signatures that enables unlinkability and selective disclosure for multi-attribute credentials.
- **CanDID** [4]: a system combining oracles, secure multiparty computation (MPC) and layered credentials to ensure Sybil-resistance, accountability and compatibility with legacy identity infrastructures.
- **Decentralized Anonymous Credentials (DAC)** [2]: a fully decentralized proposal using cryptographic accumulators and pseudonyms bound to master secrets to enforce uniqueness and support efficient revocation.
- **zk-creds** [3]: a modern system leveraging general-purpose zero-knowledge proofs (zk-SNARKs) and Merkle-based issuance to enable flexible access policies, composability and public auditability.

These systems exemplify the diverse directions in which decentralized credential research has evolved. Together, they provide a comprehensive view of current approaches and highlight the open challenges that shape the comparative focus and evaluation methodology of this thesis.

To better understand these solutions, the following sections describe the goals of each system, the properties they focus on, the mechanisms they employ to achieve them and their main limitations. The chapter concludes with a comparative analysis of all the solutions.

3.1 Coconut

3.1.1 Goal

Coconut [2] aims to eliminate the single trusted issuer that traditional credential systems depend on by distributing trust among multiple authorities through threshold cryptography. Instead of relying on one long-term signing key, Coconut assumes a set of n authorities that jointly act as the issuer under a t-out-of-n rule, so that no single authority can unilaterally issue or forge credentials. This makes Coconut one of the most influential references for threshold issuance of privacy-preserving credentials and a natural starting point for evaluating decentralized identity frameworks.

3.1.2 Properties

The protocol emphasizes three core properties:

- **Decentralization** of trust, achieved by distributing credential issuance among multiple authorities through threshold cryptography.
- **Unlinkability** between credential presentations, ensuring that verifiers cannot link multiple uses of the same credential.
- **Selective disclosure** of attributes, allowing users to prove statements about their credentials without revealing unnecessary data.

3.1.3 Mechanism

In its issuance phase, the user commits to a set of attributes and requests partial blind signatures from several authorities. Once a threshold of valid signatures is collected, they are aggregated into a single multi-attribute credential. When the credential is later presented, the user re-randomizes it and proves in zero-knowledge that it satisfies a predicate (e.g., "age ≥ 18") without revealing the underlying values. These steps collectively ensure that the credential remains unlinkable across presentations and that issuers cannot trace or correlate user activity. This design allows efficient, privacy-preserving multi-attribute credentials while giving users granular control over disclosure.

Formally, each authority returns a partial credential $\sigma_i = (h, s_i)$. Given any set S of at least t valid partial credentials, the user aggregates them with Lagrange coefficients l_i computed over the index set S:

$$\sigma \leftarrow \mathsf{AggCred}(\{\sigma_i\}_{i \in S}), \qquad \sigma = (h, s), \qquad s = \prod_{i \in S} s_i^{l_i}.$$

Under the standard threshold assumption, coalitions with $|S| < t$ cannot reconstruct the required interpolation and therefore cannot forge a valid credential [2].

From a system design perspective, Coconut's mechanism cleanly separates issuance from presentation. Threshold issuance distributes trust across authorities, while re-randomization ensures that multiple presentations of the same credential remain unlinkable. This design yields strong privacy guarantees and efficient verification, but also fixes the structure of credentials at issuance time, making later updates or policy changes costly.

3.1.4 Limitations

Despite these advantages, Coconut faces several limitations for real-world deployment. First, while a single issued credential can be re-randomized to produce an unbounded number of unlinkable presentations, the scheme does not inherently enforce uniqueness at the identity level. As a result, a user may obtain multiple root-level credentials and generate independent pseudonyms from each, undermining Sybil resistance.

This distinction between root-level credentials (issued by the authorities) and pseudonym-level presentations (derived through re-randomization) is fundamental to Coconut's privacy guarantees, but also highlights the absence of built-in mechanisms to bind credentials to unique identities.

Overall, Coconut demonstrates that decentralized threshold issuance can be both practical and efficient, as it relies on compact credentials, constant-size presentations and efficient aggregation of partial signatures. However, identity-level properties such as uniqueness and revocation are treated as external extensions rather than built-in features.

3.2 CanDID: Credential Anonymity with Decentralized Identity

3.2.1 Goal

CanDID [4] is a decentralized identity system whose goal is to issue privacy-preserving digital credentials that are both Sybil-resistant and compatible with existing real-world identity sources. It addresses the challenge of linking cryptographic credentials to verified real-world identities without compromising user privacy or introducing central points of trust. Unlike schemes that rely solely on cryptographic assumptions, CanDID allows users to bootstrap credentials from legacy data (e.g., passports, bank accounts, government databases) through trusted-oracle proofs, while still ensuring unlinkability and selective disclosure. This combination of privacy and interoperability makes it a promising candidate for practical digital identity systems.

3.2.2 Properties

The key properties that CanDID focuses on are:

- **Sybil resistance**, enforced by a deduplication step ensuring that each real person can register only once.
- **Accountability**, allowing misbehaving users to be deanonymized if necessary while maintaining privacy for honest users.
- **Selective disclosure** through zero-knowledge proofs.
- **Unlinkability** across services using context-specific pseudonyms.
- **Revocation** support.
- **Interoperability** with existing identity systems rather than requiring new trusted issuers.

3.2.3 Mechanism

CanDID achieves these goals through a two-phase credential model. In the first phase, the user obtains a master credential by submitting encrypted identity attributes to a committee of nodes running secure multiparty computation (MPC). The committee jointly verifies that the user is not already registered (deduplication) and not present on external blocklists (e.g., sanctions lists) without ever learning the user's

actual data. In the second phase, the user derives context-based credentials from the master credential, each bound to a different application and unlinkable from one another. This design prevents Sybil attacks at the root level, while allowing privacy-preserving pseudonyms for different services. Verification involves checking the committee's signature, consulting a public revocation list and validating a zero-knowledge proof that the derived credential satisfies the requested predicate. This separation between identity binding and credential presentation provides Sybil-resistance and accountability without sacrificing anonymity.

Formally, CanDID relies on a committee $C = \{C_1, \ldots, C_n\}$ that jointly holds secret shares of two keys, one for threshold signing and one for deduplication via a PRF. The committee runs distributed key generation to obtain $sk^C = (sk^C_{\text{sig}}, sk^C_{\text{prf}})$, where each node C_i holds shares $sk^C_{\text{sig},i}$ and $sk^C_{\text{prf},i}$ and the corresponding public keys $pk^C = (pk^C_{\text{sig}}, pk^C_{\text{prf}})$ are public.

To enforce uniqueness without revealing the user identifier v, the committee evaluates a deduplication token $\tilde{v}$ using secure computation:

$$\tilde{v} \leftarrow \text{PRF}([sk^C_{\text{prf}}], [v]).$$

A master credential is issued only if $\tilde{v}$ is not already present in IDTable, where IDTable is a table maintained by the committee that records PRF-derived representations of unique user identifiers, used to ensure that each identifier can be associated with at most one master credential.

Credential issuance is threshold based. Each node C_i signs the message

$$m = \{pk^U, \text{“master”}, \text{claim}, \{\text{“dedupOver”}, \{a\}\}\}$$

and produces a partial signature $\sigma^C_i \leftarrow TS.\text{Sig}(sk^C_{\text{sig},i}, m)$. After collecting t valid partial signatures, the user combines them into a full signature $\sigma^C \leftarrow TS.\text{Comb}(\{\sigma^C_i\})$ and constructs the master credential

$$cred_{\text{master}} = \{pk^U, \text{“master”}, \text{claim}, \{\text{“dedupOver”}, \{a\}\}, \sigma^C\}.$$

3.2.4 Limitations

The main drawbacks of CanDID are its operational complexity and reliance on an honest-majority committee. The use of MPC and fuzzy matching makes credential issuance computationally expensive and less scalable than purely cryptographic approaches. Moreover, unlinkability only holds as long as all the committee members behave honestly, meaning that privacy may be weakened if the committee is compromised.

3.3 DAC: Decentralized Anonymous Credentials

3.3.1 Goal

DAC [1] aim to remove the need for trusted issuers by replacing signature-based credential issuance with cryptographic accumulators and publicly verifiable logs. By doing so, it eliminates any reliance on centralized authorities and enables credentials whose validity can be verified collectively through public state rather than institutional trust. The system's goal is to support anonymous, unlinkable and revocable credentials in a fully decentralized setting, while also enforcing uniqueness so that each user can hold only one valid credential.

3.3.2 Properties

The key properties DAC focuses on are:

- **Decentralization**, since issuance and revocation rely on append-only public logs or blockchains rather than a central authority.
- **Sybil resistance**, achieved at the root level by enforcing uniqueness through cryptographic accumulators and at the pseudonym level via optional mechanisms such as k-show credentials that limit credential reuse.
- **Unlinkability**, the holder proves membership in the public accumulator using a non-interactive zero-knowledge proof of knowledge. By the zero-knowledge property, transcripts do not reveal identifying information that would allow a verifier to link presentations.
- **Efficient revocation**, by removing entries from the accumulator and publishing a new state, users update their membership witnesses accordingly, while revoked users can no longer produce valid proofs and no authority learns their identity.
- **Accountability** support through mechanisms such as k-show credentials, which allow credentials to be used only a limited number of times before revealing misuse.

3.3.3 Mechanism

DAC works as follows: during issuance, the user commits to a master secret and proves in zero-knowledge that this secret has not already been included in the accumulator, ensuring uniqueness without revealing identity. The accumulator acts as a global and publicly verifiable registry of valid users, allowing verifiers to efficiently check credential validity through short membership proofs.

When showing a credential, the user produces a zero-knowledge proof of membership in the accumulator, optionally combined with additional statements such as k-show constraints or pseudonyms derived from the same master secret. Each presentation is generated with fresh randomness and, by the zero-knowledge property, does not reveal information that would allow verifiers to correlate multiple presentations.

Revocation is performed by updating the accumulator and publishing a new state. Affected users must refresh their membership witnesses, while revoked credentials can no longer produce valid proofs. This process does not reveal which user was revoked, since only the updated accumulator state is made public. The model therefore decouples trust from cryptographic validity: correctness is enforced mathematically, while transparency and policy enforcement come from the public ledger, understood here as an append-only public log that records accumulator updates and can be independently verified.

Formally, DAC represents global membership using a Strong RSA accumulator with public parameters (N, u). Let $C = \{c_1, \ldots, c_n\}$ be the set of accumulated values, where each c_i is a prime in a public range. The accumulator value is

$$A = u^{\prod_{j=1}^{n} c_j} \bmod N.$$

A user associated with c_i holds a witness

$$\omega_i = u^{\prod_{j \neq i} c_j} \bmod N,$$

and membership is verified by checking

$$A \equiv \omega_i^{c_i} \pmod{N},$$

together with the requirement that c_i is prime and lies in the allowed range [1].

To preserve anonymity, the holder does not send c_i and ω_i in the clear. Instead, the presentation includes a non interactive zero-knowledge proof of knowledge showing that the holder knows a value c and a witness ω that satisfy the accumulator relation:

$$\mathrm{NIZKPoK}\{(c, \omega) : A \equiv \omega^c \pmod{N} \ \wedge \ c \text{ is prime in the required range}\}.$$

This allows the verifier to check membership against public state while the holder keeps identifying information hidden.

3.3.4 Limitations

The main limitations of DAC stem from its reliance on public accumulator state. Frequent revocations require non-revoked users to refresh their membership witnesses, increasing communication and computation overhead over time. Verification also depends on the latest accumulator value: verifiers must obtain a current accumulator state from the public log and check the associated membership proof (or the zero-knowledge proof that includes it). As a result, both holders and verifiers must remain online or periodically synchronize with the ledger, which can pose practical challenges for scalability, latency and user experience in large, distributed deployments.

3.4 zk-creds

3.4.1 Goal

zk-creds [3] is an anonymous credential system that introduces flexibility in how credentials are issued, used and verified. It replaces fixed-purpose credential logic with general purpose zero-knowledge proofs, enabling dynamic and composable access policies defined even after issuance. The system follows an issuer-agnostic model, avoiding trusted signing keys and treating issuance as the insertion of a credential into an auditable public list (e.g., bulletin board, transparency log or blockchain). Each credential is represented as a leaf in a Merkle tree, which supports verifiable inclusion proofs, publicly verifiable revocation and efficient scalability.

3.4.2 Properties

The main properties zk-creds focuses on are:

- **Flexibility**, through zero-knowledge gadgets that define and combine access rules dynamically.
- **Issuer-agnostic design**, since no entity must store signing keys, anyone can issue as long as the list is public and append-only.
- **Public auditability**, with all issued credentials verifiable through the public list.
- **Unlinkability**,achieved because presentations are zero-knowledge proofs generated with fresh randomness and do not reveal stable identifiers that would allow verifiers to correlate multiple uses.
- **Selective disclosure** of attributes via ZK circuits.
- **Sybil resistance (configurable)**, which can be enforced at the root level through uniqueness proofs or issuance restrictions and at the pseudonym level through circuit-defined constraints such as k-show limits or pseudonym bindings.
- **Efficient revocation** by removing a credential's leaf from the Merkle structure at logarithmic cost.

3.4.3 Mechanism

In practice, zk-creds issuance begins when a user commits to a set of attributes and inserts this commitment into a Merkle tree that represents the public issuance list. Optionally, the user may attach additional zero-knowledge proofs to demonstrate the authenticity of those attributes (e.g., showing that they match a passport without revealing it). When presenting a credential, the user produces a single zk-proof that jointly proves two statements: (1) the credential is still included in the public issuance list (membership proof) and (2) its attributes satisfy the verifier's predicate, expressed through reusable gadgets such as age checks, pseudonym bindings or k-show limits. The use of zero-knowledge SNARK proofs allows the holder to prove policy compliance without revealing the underlying credential data. Revocation is handled via updates to the public list, represented as a Merkle structure. Once a credential is revoked, verifiers check presentations against the latest Merkle root, and proofs tied to revoked entries will no longer verify. Holders may need to update their Merkle authentication path (witness) as the tree evolves.

Formally, zk-creds represents a credential as a commitment to a set of values:

$$\mathsf{cred} := \mathsf{Com}(nk, rk, \mathsf{attrs}; r),$$

where nk is a secret pseudonym key, rk is a secret rate key, attrs is the attribute set and r is the commitment randomness.

A credential is issued when it is added to a public issuance list. In the main instantiation of zk-creds, this list is represented as a Merkle forest. The credential appears as a leaf in a Merkle tree T with root T_{root} and the holder stores an authentication path θ as a witness that $\mathsf{cred} \in T$. The forest F contains the set of accepted tree roots.

To show a credential, the holder provides a proof π that it knows the committed values $(nk, rk, \mathsf{attrs}, r)$ together with a valid Merkle witness θ proving inclusion of cred in T. The same proof also shows that T_{root} belongs to the accepted roots in F. This shifts trust away from a single issuer key and toward the integrity of public state [3].

3.4.4 Limitations

zk-creds' limitations are mainly practical rather than cryptographic. Since credentials are stored in Merkle trees, users must periodically update their membership witnesses, introducing synchronization overhead. The system also relies on a trusted setup for Groth16 proofs, which introduces an explicit trust assumption typically addressed through multi-party setup ceremonies. Finally, supporting arbitrary access policies through general purpose circuits increases computational cost, illustrating a trade-off between flexibility and efficiency. Despite these challenges, zk-creds remains one of the most auditable and expressive credential architectures available.

3.5 Comparative Analysis

To better understand the trade-offs and strengths of current approaches to decentralized anonymous credential systems, we compare the core properties of the four representative works: CanDID, Coconut, DAC and zk-creds.

System / System Elements	CanDID	Coconut	zk-creds	DAC
Issuance model	Committee (MPC) + oracles (DECO/Town Crier); master credential + context credentials	Threshold issuance with blind signatures	Issuance via auditable list (Merkle/BB/chain) or by signatures; gadgets for access policies	Decentralized issuance with accumulators and pseudonyms bound to a master secret
Decentralization / governance	Committee-based MPC; threshold trust in committee, plus reliance on external legacy identity providers (via oracles)	Set of authorities (threshold t-of-n)	Public list (BB/chain) or committee; highly flexible	Ledger/transparency log + multiple entities
Sybil-resistance	Yes (strong) through deduplication (unique IDs + PRF in MPC)	No native support (multiple credentials possible)	Configurable: via fees, email DKIM, signed docs $\rightarrow$ yes if enforced	Yes: uniqueness via accumulators and pseudonyms linked to a secret
Unlinkability	Across contexts (pairwise DIDs)	Strong (re-randomization of credentials)	Default unlinkability (optional controlled linkability)	Strong (supports k-show but configurable)
Selective disclosure	Yes (via contexts and ZK)	Yes (multi-attribute, ZK proofs)	Yes (general-purpose SNARKs, gadgets)	Yes (classic in anonymous credentials)
Revocation	Supported (lists/attributes, MPC; requires careful handling)	No (not built in)	Efficient (removal in Merkle tree; $O(\log N)$)	Efficient with accumulators (membership/non-membership proofs)
Accountability	Yes (fuzzy matching with sanctions lists in MPC)	No (not built in)	Public auditability of issued credentials	Auditability via log/ledger
Legacy identity compatibility	Strong (oracles to legacy web services)	Not focused on legacy IDs	Strong (e.g., passports with zk-support)	Possible; more generic
Public auditability	Limited (committee-level)	No (signatures only)	Yes (issuance list / transparency log)	Yes (log + accumulators)

Table 3.1: Comparison of CanDID, Coconut, zk-creds and DAC along core system properties.

All four systems pursue the same goal of providing privacy-preserving digital identities, but each adopts a different balance between decentralization, security and efficiency.

A first contrast lies in the **issuance model**. Coconut distributes authority through threshold signatures, while CanDID uses a committee with secure multiparty computation, introducing stronger verification guarantees at higher cost. DAC replaces signatures altogether with cryptographic accumulators, achieving full decentralization but requiring continuous state synchronization. zk-creds builds upon this model by generalizing DAC's public-log-based issuance and verification through general-purpose zero-knowledge proofs, enabling composable and auditable credential logic.

In terms of **Sybil resistance**, two different levels can be distinguished. At the root level, CanDID and DAC enforce uniqueness at registration time, ensuring that each individual can hold only one valid root credential. CanDID achieves this through a deduplication step in MPC, while DAC relies on cryptographic accumulators that prevent the same secret from being registered twice. In contrast, Coconut and zk-creds lack inherent root-level Sybil protection, although zk-creds can incorporate external enforcement mechanisms, such as issuance fees or unique attestations, to approximate this property. At the pseudonym level, systems like CanDID and DAC maintain unlinkable yet bounded pseudonyms derived from a single root identity, whereas Coconut and zk-creds provide full unlinkability across contexts without built-in constraints on multiple pseudonym creation.

Privacy mechanisms vary across implementations. Coconut relies on blind signatures and credential re-randomization, yielding strong unlinkability with relatively lightweight presentations. DAC derives pseudonyms from a master secret and relies on zero-knowledge membership proofs over public state, achieving unlinkability at the cost of additional operational overhead due to state synchronization. zk-creds employs general purpose zero-knowledge SNARKs to express flexible predicates at presentation time, providing the highest expressiveness but incurring higher proving and verification costs. While all

approaches aim for unlinkability, they realize it through different mechanisms and make distinct trade-offs between expressiveness and computational efficiency.

Finally, **revocation** mechanisms highlight major architectural contrasts. Coconut does not include revocation as a built-in primitive. CanDID supports revocation through committee-managed lists, which enable fine-grained control but introduce additional operational complexity. By contrast, DAC and zk-creds achieve verifiable revocation through public-state updates. DAC relies on accumulator-based membership proofs, while zk-creds updates a public credential list represented as a Merkle structure. In both systems, revocation is publicly observable at the level of credential entries, while the holder's real-world identity need not be revealed. zk-creds further emphasizes auditability by recording issuance and revocation events on a public log, allowing changes to the credential set to be publicly verified.

Overall, the analysis shows that no system fully satisfies all desirable properties. Coconut excels in decentralization and efficiency but lacks strong identity guarantees. CanDID prioritizes accountability and real-world integration at the cost of operational complexity. DAC achieves strong uniqueness and decentralization but sacrifices attribute flexibility. zk-creds provides maximum expressiveness and auditability, although with higher computational overhead.

This comparison suggests that a promising research direction is not to replace one system with another, but to **combine complementary mechanisms** for example, integrating zk-creds' gadget-based access control or DAC-style accumulators into Coconut to enable Sybil resistance and efficient revocation, while preserving its decentralized issuance and strong unlinkability. In particular, Chapter 6 explores this direction by investigating how Coconut could be augmented with Sybil-resistance and controlled re-use detection, inspired by the kind of circuit-defined constraints ("gadgets") used in zk-creds. Such modular combinations motivate the exploration carried out in the rest of this thesis.

3.5.1 From Qualitative Comparison to Quantitative Evaluation

The comparison in this chapter shows that decentralized anonymous credential systems differ far beyond their cryptographic primitives. They also make distinct choices about coordination, public state and governance. As a result, qualitative arguments alone are often insufficient to decide which approach fits a concrete deployment scenario.

A key limitation of the literature is that performance results are rarely directly comparable. Implementations use different parameter choices, hardware environments and measurement boundaries and they often benchmark different parts of the lifecycle. Moreover, decentralization introduces costs that are easy to underestimate from a design description: committee-based issuance can add interaction rounds and latency, while approaches relying on public state may shift complexity to maintaining and querying registries or logs.

For these reasons, this thesis complements the qualitative comparison with a controlled benchmark. The evaluation focuses on a small set of metrics that capture the main practical bottlenecks: issuance time, verification time and credential/proof size. This empirical perspective helps ground the trade-offs discussed in the state of the art and supports more informed engineering choices.

4
Project Development

The empirical part of this thesis lives and dies by one question: can the numbers be trusted? Benchmarking cryptographic systems is deceptively fragile. A small change in measurement boundaries or a missing warm-up phase can easily dominate the result. For this reason, the benchmark was developed as a dedicated benchmarking infrastructure designed to be easy to inspect, reproduce and extend. The goal is not only to produce plots, but to leave behind an artifact that a reader can examine, re-run and adapt.

The benchmark focuses on Coconut and zk-creds. These two systems were selected because they provide publicly available research implementations with clearly defined issuance, presentation and verification workflows that can be executed in isolation. In contrast, CanDID and DAC rely on additional system components, such as MPC committees or continuously synchronized public ledgers, that are integral to their design but difficult to model faithfully in a controlled benchmarking setting. Evaluating those systems would require introducing substantial deployment assumptions beyond the scope of this work, potentially obscuring the cost patterns under study.

At a high level, this benchmarking infrastructure performs three functions. It runs Coconut and zk-creds in their native environments, records raw measurements in a consistent format and produces the processed datasets and figures used in Chapter 5. The sections that follow describe the design decisions that make these measurements comparable and reproducible, and point to the appendix where the exact timing and size accounting code is documented.

4.1 Design Goals

The benchmark was built around three goals.

1. Every result should be reproducible. That means the pipeline must be executable end-to-end with a small set of commands and it must capture enough metadata to reconstruct the exact experimental context later on. In practice, this includes commit hashes, OS and hardware details and tool versions.

2. Even when the internal structure of the schemes differs, the benchmark should expose a common set of quantities that match the credential lifecycle. We measure time for `issue`, `show` and `verify` and we record size estimates that reflect what a holder stores and what a verifier must receive in order to validate a presentation.

3. The benchmark should be easy to extend. The intent is not to freeze a one-off experiment, but to leave a structure where new scenarios or additional systems can be added without rewriting everything from scratch.

These goals lead to a simple architecture: each scheme produces per-run measurements in a scheme-specific CSV file and a small aggregation step merges them into a single processed dataset for plotting and analysis.

4.2 Benchmark Architecture

A key design choice in this project is to avoid forcing the evaluated systems into a shared implementation interface. Coconut and zk-creds follow different design philosophies, build processes and abstractions, and rewriting them behind a common API would introduce an additional layer of complexity that is difficult to validate and easy to misinterpret. Instead, the benchmarking infrastructure standardizes the experimental context around each system: how experiments are executed, how repetitions and warm-up phases are handled, how measurements are recorded and how results are aggregated into figures.

For both systems, measurements are performed by executing the full issuance and presentation lifecycle using the native implementations provided by each project. Although the measurement logic is system-specific, both benchmarks produce the same categories of output, namely per-run timing measurements and size estimates, which are later processed in a uniform way.

The repository is organized in three logical layers. External implementations contain the upstream codebases of the evaluated systems. The benchmarking layer contains the orchestration logic, scripts and aggregation code. Finally, the results layer stores all generated artifacts, including raw CSV outputs, the processed datasets used for figures and metadata capturing the execution environment and pinned code revisions.

```
TFM-CREDENTIAL-BENCHMARK/
  external/                 # upstream repos
    coconut/
    zk-creds-rs/
  harness/                  # benchmark logic
    run_coconut.sh          # runner: executes coconut_quick_bench.py
    coconut_quick_bench.py  # coconut benchmark
    run_zk-creds.sh          # runner: applies zk-creds patch
    patches/
        zk-creds_bench.patch # zk-creds benchmark
    aggregate_results.py    # merges raw CSVs into a unified dataset
    plot_results.py         # produces static plots
    gui_compare.py          # optional Streamlit inspection
    system_info.sh          # environment metadata
    versions.sh             # versions of key tools and dependencies
    commits.sh              # Git commit hashes
  plots/                    # generated figures
  results/                  # all reflecting artifacts
    metadata/               # system_info.txt, versions.txt, commits.txt
    raw/                    # scheme-specific CSVs
      coconut/
      zk-creds/
    processed/              # aggregated CSV used for plotting
```

Figure 4.1: High-level benchmark layout.

Execution flow. Each benchmark run begins by recording metadata under `results/metadata/`. This step captures the execution context before any measurements are taken, so that every figure can later be traced back to a concrete setup. The benchmark then runs Coconut and zk-creds under the same warm-up and repetition discipline and stores per-run measurements in `results/raw/`. Raw results are kept at per-run granularity on purpose. It allows us to compute robust statistics later on and to inspect variance rather than hiding it behind averages. Finally, an aggregation script merges the raw CSV files into a unified dataset under `results/processed/` and a plotting script generates the figures included in the thesis under `plots/`. For convenience during development, the benchmark also provides an optional GUI that allows quick filtering and distribution-level inspection.

The pipeline is exposed through Makefile targets that map directly to these stages. This reduces configuration drift and makes it straightforward to regenerate the full set of results (Figure 4.2).

```
8   help:
9       @echo "Targets:"
10      @echo "  setup          Create venv + install deps (if needed)"
11      @echo "  metadata       Collect reproducibility metadata into results/metadata/"
12      @echo "  bench          Run full pipeline: metadata + coconut + zkcreds + aggregate"
13      @echo "  bench-coconut  Run only Coconut benchmark"
14      @echo "  bench-zkcreds  Run only zkcreds benchmark"
15      @echo "  aggregate      Build results/processed/bench_results.csv"
16      @echo "  plot           Generate plots/summary under plots/"
17      @echo "  gui            Launch interactive Streamlit UI"
18      @echo "  clean          Remove generated artifacts (results/* and plots/*)"
19      @echo "  all            setup + bench + plot"
```

Figure 4.2: Makefile entry points for the benchmark (output of `make help`).

4.3 Implementation of the Coconut benchmark

The Coconut benchmark is a Python script that executes a complete lifecycle iteration: issuance, presentation generation and verification. Rather than measuring a single configuration, the script sweeps a small grid by varying the total number of attributes A, the number of revealed attributes R and the committee parameters (t, n). The implementation is based on the Coconut reference code [5].

To avoid mixing initialization cost into per-credential measurements, cryptographic setup is performed once per benchmark invocation and excluded from timing. This includes generating global parameters, ElGamal keys for blinding and unblinding, threshold authority keys and the aggregated verification key used by verifiers.

Each measured iteration is then split into three timed experiments. The `issue` region covers blind issuance preparation, threshold signing, unblinding and aggregation into the final credential σ. The `show` region measures presentation generation, producing the generated proof Θ, from σ, through `prove_cred`. The `verify` region measures verifier-side checking through `verify_cred`. Timings are recorded using `time.perf_counter` and each scenario includes warm-up iterations that are executed but excluded from the recorded CSV. The exact timing boundaries are shown in Appendix A (Figure A.1).

Size accounting for Coconut requires a proxy, because the implementation does not expose canonical serialization for all internal objects. The benchmark therefore uses a best-effort strategy that prefers byte exports when available and otherwise falls back to structured recursion for lists and dictionaries. The logic is made explicit in Appendix A (Figure A.2). The script records the stored credential size, presentation size, public input size and a verifier payload estimate computed as $|\Theta| + |public\_m|$. Raw results are written to `results/raw/coconut/quick_bench.csv`.

4.4 Implementation of the zk-creds Benchmark

The zk-creds benchmark is implemented as a Rust test that measures a concrete end-to-end flow based on the existing `RevealingMultishowChecker` example instantiated over the `NameAndBirthYear` attributes [6]. This is the only flow available in the codebase that fits naturally into the same measurement framing as Coconut, because it resembles a show/prove plus verify interaction and includes verifier-facing artifacts beyond the proof itself.

To avoid maintaining a long-lived fork of the upstream repository, the benchmark is kept as a small Git patch stored under `harness/patches/` and applied automatically during execution. This keeps the upstream codebase intact while making the benchmark additions explicit and reviewable.

As with Coconut, each measured iteration is split into three timed regions. The `issue` region here corresponds to per-credential materialization in the evaluated flow, namely generating the multishow token and committing to the attributes. This is not issuance in the threshold-signature sense, but it captures the local setup work required before a holder can generate a proof. The `show` region measures SNARK proof generation for the selected predicate. The `verify` region measures verifier-side proof checking against the commitment and verifier-facing inputs. Timings are recorded using `std::time::Instant`, warm-up iterations are excluded and the exact timing boundaries are shown in Appendix A (Figure A.3 and Figure A.4).

Unlike Coconut, zk-creds supports canonical serialization, so size estimates can be computed as serialized byte lengths. The benchmark records the size of the stored credential material, the proof, the verifier-facing public inputs and an estimated verifier payload. The size accounting code is shown in Appendix A (Figure A.5). Raw results are written to `results/raw/zkcreds/quick_bench.csv`.

4.5 Reproducibility Metadata

To make results traceable, the benchmark records a small set of metadata under `results/metadata/`. It includes pinned commit hashes for the benchmark and external dependencies, system information about the machine used for evaluation and toolchain versions for Python and Rust. For example, the benchmark records the exact Git commit hashes for itself and for each external dependency, so every plot can be traced back to a precise code state (Figure 4.3). This metadata is captured automatically as part of the pipeline, so it stays consistent with the produced CSV outputs and plots.

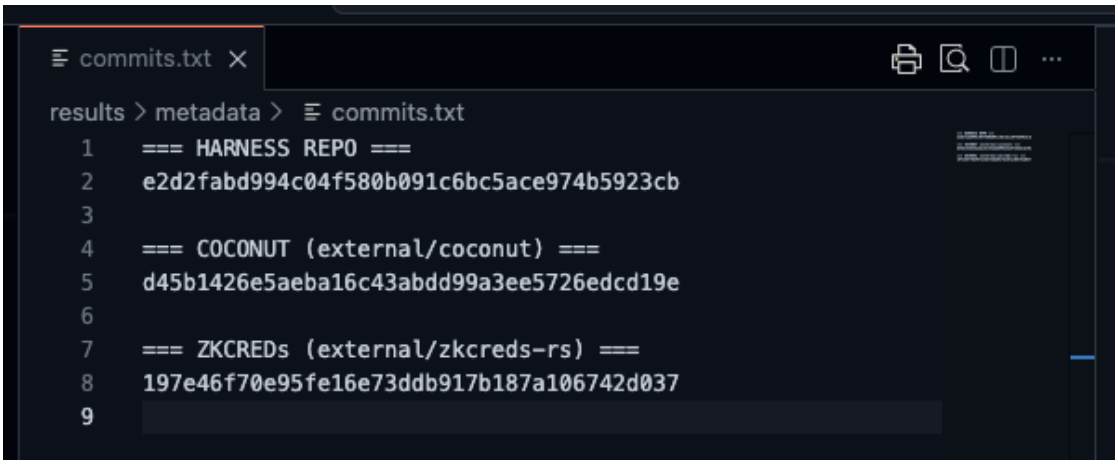

Figure 4.3: Recorded commit hashes (`results/metadata/commits.txt`).

5 Experimental Evaluation

This chapter reports the empirical comparison between **Coconut** and **zk-creds** using the benchmark described in Chapter 4. The goal is not to derive a general ordering of systems, but to make a single, well-scoped question measurable: when the same credential lifecycle is executed end-to-end, where does each scheme incur time and payload cost?

A recurring theme throughout this chapter is that comparability is a design choice. Coconut and zk-creds do not expose the same scenario space in their available implementations. Rather than forcing a misleading comparison, we make the boundaries explicit and focus on the only point where the benchmark can be strict.

5.1 Experimental design

5.1.1 Measured metrics

A credential system is experienced through its lifecycle. For that reason, the benchmark measures the three stages that directly translate into user and system cost.

For each run, we record the following values:

- `issue_ms`: holder-side work required to obtain usable credential material for this flow, measured from the start of issuance or credential materialization until the holder has everything needed to later execute `show`. One-time setup is excluded.
- `show_ms`: holder-side time to generate a presentation or proof that satisfies the verifier policy.
- `verify_ms`: verifier-side time to check the received presentation and decide whether to accept or reject the holder's claim.

Alongside time, the benchmark also measures sizes that reflect operational constraints:

- `credential_size`: number of bytes stored by the holder as credential material after `issue` in this benchmark flow.

- `network_verify_size`: number of bytes the verifier must receive and process to complete verification in this benchmark flow. In our accounting, this includes the presentation or proof together with verifier-facing inputs required by the protocol.

All plots use the **median** across repeated runs after excluding warm-up iterations. Medians are used because they remain stable under occasional noise from scheduling, caching and rare slow runs.

5.2 Scenario

The benchmark compares Coconut and zk-creds on the only credential flow that is available in both implementations in a directly compatible form. In this scenario, the credential contains two attributes and the holder reveals none of them to the verifier. The interaction is therefore a privacy-style presentation: the verifier learns that the holder satisfies the policy and that the presentation is valid, without learning the underlying attribute values.

With revealed attributes $R = 0$, meaning that no attribute values are disclosed, but the holder still proves in zero-knowledge that it controls a valid credential issued under the correct parameters and that the resulting presentation or proof is well formed and verifies.

In zk-creds specifically, the proof shows that the predicate holds for the committed attributes, such as a birth-year check, without revealing the attributes themselves.

Considerations.

- **Why this scenario.** zk-creds is benchmarked through the circuit example available in the codebase, `RevealingMultishowChecker`, which fixes the evaluated flow. Coconut can be used in multiple configurations, but only this flow can be matched across both schemes without introducing additional assumptions.
- **What is being compared.** The comparison captures the cost profile of a presentation where validity and policy compliance are proven while attribute values remain hidden.
- **Scope of coverage.** Coconut is also evaluated over a grid by varying the total number of attributes and the number of revealed attributes. Those additional points are used only to show within-scheme trends for Coconut, not to support cross-scheme scaling claims.

5.3 Results and interpretation

5.3.1 Runtime Results

Figure 5.1 shows the median execution time of each lifecycle stage for Coconut and zk-creds under a comparable flow. Rather than only comparing absolute runtimes, the figure highlights how computation is distributed across stages. Coconut concentrates most of its cost in the issuance phase, where blind signing and aggregation dominate, while presentations remain relatively lightweight. In contrast, zk-creds shifts the computational burden to the presentation phase, as generating zero-knowledge SNARK proofs is substantially more expensive, while issuance mainly consists of inserting entries into a public list. As a result, each system exhibits a distinct performance profile shaped by the stage it optimizes for.

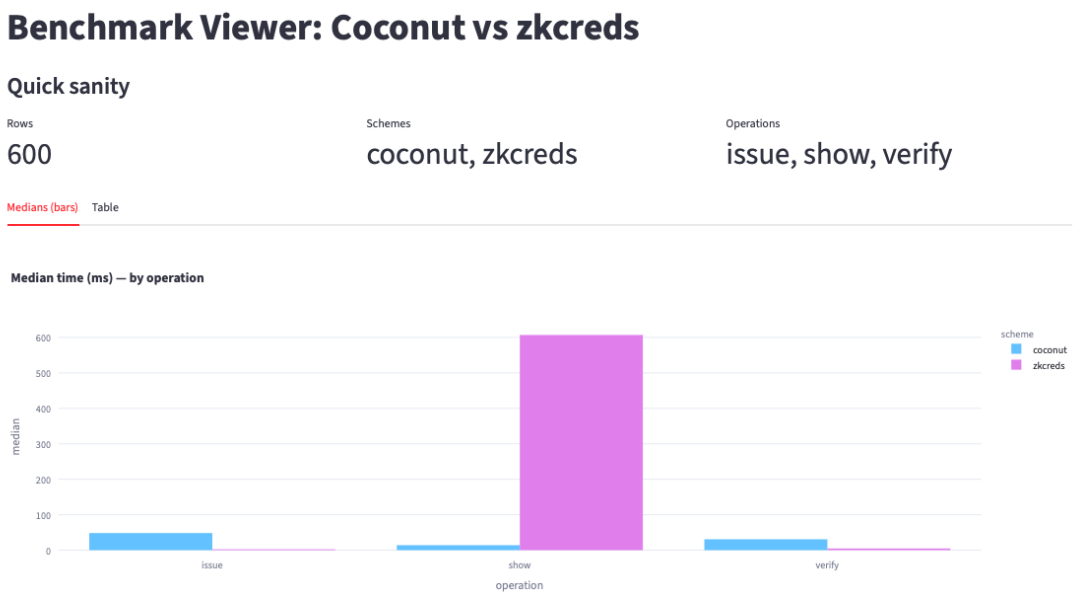

Figure 5.1: Median time by lifecycle stage for Coconut and zk-creds under the comparable flow.

Issue. Coconut spends noticeably more time in `issue` than zk-creds, on the order of a few tens of milliseconds in this scenario. Concretely, the median issuance time is around 48 ms for Coconut, compared to approximately 0.6 ms for zk-creds (see Appendix B). This difference is expected because Coconut uses threshold issuance. Even in the minimal threshold case, issuance is not a single local step: the holder prepares a blinded request, multiple authorities compute partial signatures, the holder unblinds them and finally aggregates them into a usable credential. The benchmark counts exactly this per-credential work, so the extra cost reflects the committee structure and the signature aggregation logic.

In zk-creds, the measured `issue` for this flow is closer to local credential preparation. The holder computes the multishow token and commits to the attributes. There is no threshold signing step in this benchmarked path, which explains why the median `issue` time remains below one millisecond.

Show. The largest gap appears in `show`. zk-creds is roughly two orders of magnitude more expensive in this stage, with median proving times in the range of 650 ms for this flow (see Appendix B). This cost is dominated by the generation of a SNARK proof: the holder must satisfy the full constraint system of the circuit and produce a general-purpose zero-knowledge proof, which is an explicit design choice in SNARK-based systems.

Coconut, by contrast, keeps `show` lightweight in this configuration, with median times around 14 ms. Once the threshold-issued credential has been produced, generating a presentation is closer to deriving a compact proof of possession from an already formed credential than to executing a large general-purpose circuit. This is where Coconut amortizes the higher cost paid during issuance.

Verify. Verification follows a similar pattern to `show`, but with much smaller absolute costs. zk-creds verification is fast by construction, with median verification times around 4 ms in this benchmark (see Appendix B), as SNARK verification is designed to be efficient once the proving and verification keys are fixed. Coconut verification is slightly higher, with median times around 32 ms, since the verifier must validate the aggregated threshold verification key and check protocol-specific relations that bind the

presentation to the public inputs. Although the absolute gap is modest compared to `show`, it consistently reflects the additional checks performed during Coconut verification.

Interpretation. Taken together, the timing results highlight a clear difference in where computational cost is incurred across the credential lifecycle. Coconut incurs higher cost during `issue`, where threshold signing and aggregation are performed, and keeps `show` relatively lightweight. By contrast, zk-creds shifts most of the computational burden to `show`, as generating a zero-knowledge SNARK proof dominates the cost, while `verify` remains inexpensive by construction.

As a result, the relative suitability of each approach depends on where a deployment can afford its dominant cost: during issuance, during user-facing proof generation, or during large-scale verification. This trade-off is central to choosing an architecture that aligns with practical deployment constraints.

5.3.2 Size Results

Figure 5.2 compares the median size of what the holder stores and what the verifier must receive to complete verification.

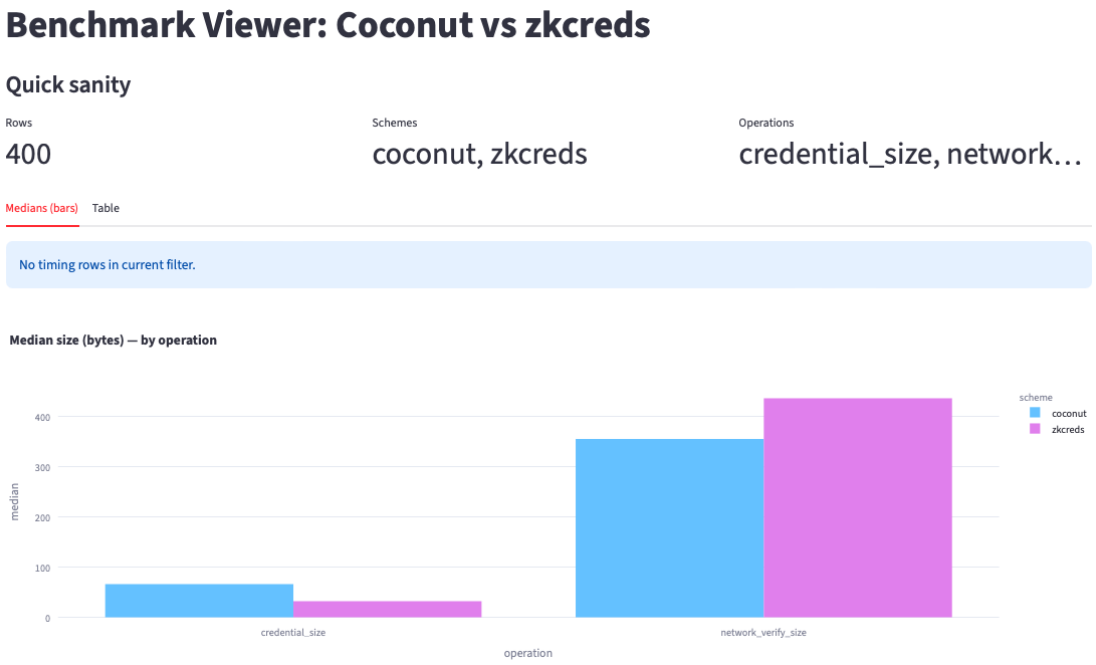

Figure 5.2: Median size metrics for Coconut and zk-creds under the comparable flow.

Stored credential size. In this flow, zk-creds stores a compact credential artifact, essentially a commitment to the attributes together with secret material used later for proving. This results in a stored credential size of 32 bytes in the benchmark (see Appendix B). Coconut stores a credential object that includes a signature over multiple attributes, resulting in a larger stored credential size of 66 bytes.

Although threshold issuance is used, the aggregated threshold signature in Coconut is comparable in size to a single signature of the same scheme. The observed difference in credential size therefore does not stem from threshold aggregation itself, but mainly reflects the underlying credential representation

and the fact that signature-based credentials carry group elements, which tend to be heavier than a single commitment.

Verification payload. The direction flips when looking at network-facing size. In this benchmark, zk-creds produces a larger verification payload of 436 bytes, compared to 355 bytes for Coconut (see Appendix B). This difference stems from the structure of SNARK-based verification: the payload must include the proof itself together with all public inputs required for verification, such as the current Merkle root, index or path information and policy-related parameters. Even when no attributes are disclosed, these public inputs are necessary to make the proof verifiable.

Coconut is smaller in this benchmark, as the verifier receives a compact presentation together with the public inputs of the flow. As a result, the verification payload is closer to a lightweight proof of possession than to a full SNARK proof package.

Operational implications. While the absolute differences are on the order of a few hundred bytes, payload size still matters in practice. It affects bandwidth consumption, end-to-end latency over constrained links and batching efficiency when a verifier processes many presentations. The size results therefore reinforce the same message as the timing results: Coconut tends to minimize show-time payloads, while zk-creds trades larger proof objects for fast verification and circuit expressiveness.

5.3.3 How this translates to deployments

The benchmark highlights two different ways of "paying" for privacy. Coconut concentrates work in issuance so that later presentations remain relatively smooth, while zk-creds concentrates work in proving so that verification remains extremely cheap.

In a deployment, this difference maps to very concrete operational questions. If a credential is shown repeatedly, the cost of `show` becomes the user-facing bottleneck. Coconut's profile fits settings where holders present often and latency during showing matters, because the system has already invested effort during issuance to produce a ready-to-use credential. This can be attractive for membership-style credentials, recurring access decisions or any workflow where a single issuance supports many future presentations.

zk-creds fits a different pressure point. When verification happens at scale, the verifier becomes the system's choke point. zk-creds keeps `verify` small by design and shifts most effort to proof generation. This profile aligns with verifier-heavy environments, such as gates, infrastructure access control or large services that must validate many proofs quickly. The trade-off is that holders must tolerate higher `show` latency and heavier local computation.

Size results reinforce the same story. zk-creds stores a smaller credential artifact in this flow, which is helpful for wallet-like storage. At the same time, its verifier payload is larger because the proof package and verifier inputs must travel with each presentation. Coconut's payload is smaller here, which can help in bandwidth-constrained settings or when many presentations are transmitted and processed.

Overall, the comparison suggests a simple deployment rule: choose the scheme that places the dominant cost where the system can afford it. Coconut is attractive when showing must be fast and smooth for the holder. zk-creds is attractive when verification throughput and policy expressiveness dominate, accepting that proving can be the expensive step.

5.4 Scope and Validity of the Conclusions

The benchmark supports a small set of concrete conclusions for the evaluated flow. Coconut concentrates more cost in `issue` while keeping `show` comparatively cheap, whereas zk-creds concentrates most

of its cost in `show` while keeping `verify` inexpensive. On the size side, zk-creds stores a smaller credential artifact in this benchmark, but produces a larger verifier-facing payload than Coconut under the measurement definitions used here.

These conclusions are intentionally scoped. They describe the cost profile of the specific implementations and measurement model exercised by the benchmark, and should not be generalized beyond that context.

First, the measurements reflect concrete toolchains and integration choices. Coconut is evaluated via Python bindings, while zk-creds is evaluated in Rust. The goal is to provide reproducible engineering evidence rather than isolate language-level performance.

Second, zk-creds performance is inherently circuit-dependent. The reported profile corresponds to the evaluated circuit and access policy, and different predicates or constraint systems may significantly shift the balance between `show`, `verify` and payload size.

Third, size accounting is not symmetric across schemes. zk-creds sizes are derived from canonical serialization of proofs and public inputs, while Coconut sizes are best-effort proxies. Here, a best-effort proxy refers to an approximate size estimate derived from the data exchanged during verification, rather than from a canonical serialization of the credential or proof object itself. For this reason, cross-scheme size comparisons are interpreted primarily through network-facing size, which captures the verifier-facing payload under the benchmark's accounting model.

Finally, the benchmark does not attempt to scale across a wide range of zk-creds configurations. Extending the evaluation to additional policies or flows would require the design, implementation and validation of new zero-knowledge circuits, which is a non-trivial engineering effort and would introduce additional sources of variability. To preserve comparability and reproducibility, the benchmark therefore focuses on a single representative flow rather than a broad but heterogeneous set of circuits.

6
Exploratory Work

Our comparative analysis highlighted where Coconut is particularly strong and where it is comparatively weaker than other anonymous credential families. Coconut stands out for threshold issuance, compact credentials and efficient pairing based verification, while still providing strong anonymity and unlinkability across verifiers. At the same time, several deployment oriented features appear more naturally in other designs. This chapter documents exploratory work motivated by the following question:

> *If Coconut is our baseline, which ideas from other models could plausibly be adapted to Coconut without sacrificing its core properties?*

Concretely, we explored three directions inspired by the comparative study:

- **Local Sybil resistance:** inspired by zk-creds and its intuition around cloning resistance and multi show control.
- **Revocation:** inspired by accumulator based approaches seen in DAC style systems. inspired by accumulator-based approaches seen in DAC [1] where construction use public accumulators and membership witnesses to enable revocation.
- **Operational overhead of updates:** the fact that changing a single attribute can require full re issuance, increasing operational costs.

Among these, local Sybil resistance led to the most concrete technical experiment, so we present it in detail, followed by shorter notes on revocation and update costs.

6.1 Sybil Resistance Attempts: Verifier-Local Reuse Detection

This section documents an exploratory attempt to add *verifier-local* reuse detection to Coconut without sacrificing what makes Coconut attractive in the first place: anonymity, unlinkability across verifiers and threshold issuance. The motivating tension is simple to state but tricky to resolve.

In Coconut, a verifier can check that a presented credential is valid and that the user satisfies a policy, yet the verifier does not learn who the user is. This is exactly the point of anonymous credentials, but

it also creates a practical headache in deployments that resemble access control. A service may want to enforce rules such as one access per user, one vote per user or one redemption per ticket. If the verifier cannot tie repeated presentations to the same underlying credential, it becomes hard to detect reuse and, more subtly, hard to discourage credential lending. A user could present access on behalf of someone else who is authorized and the verifier would have no stable handle to notice.

zk-creds provides a useful mental model for this tension. In zk SNARK based designs, one can sometimes introduce controlled linkability using circuit defined gadgets, so that a single verifier can detect repeated use while the user remains unnamed. This naturally suggested the question that guided our exploratory work:

> *Can we give Coconut a verifier-local signal that two re-randomized presentations come from the same credential, while keeping the user anonymous and keeping different verifiers unlinkable?*

The difficulty is that Coconut enforces unlinkability by construction. Credentials are re-randomized before each show, so the signature representation does not provide a stable identifier. Any reuse detection must therefore come from something other than the raw credential representation.

6.1.1 A first idea: a verifier local tag derived from the signed exponent

Coconut credentials build on Pointcheval Sanders signatures [7]. In our notation, a credential in a multi attribute setting can be written as

$$\sigma = (h, s), \qquad s = h^{x+y_a \cdot a + y_w \cdot w},$$

where a is a regular attribute, w is a secret attribute intended to behave like a pseudonymous handle and (x, y_a, y_w) are secret key coefficients or, in the threshold case, aggregated contributions derived from shared coefficients.

It is convenient to name the exponent

$$m = x + y_a \cdot a + y_w \cdot w.$$

For completeness, the implementation details and extended discussion of this conclusion are included in Appendix C.

The temptation is then to define a verifier specific tag as

$$\text{tag}_v = H(\text{id}_v)^m,$$

where H hashes a verifier identifier id_v into the appropriate group. Intuitively, this looks like it could give exactly what we want. For a fixed verifier v, the tag would be deterministic across multiple shows of the same credential, yet m would not be revealed due to the hardness of discrete logarithms. In other words, a verifier could spot reuse without learning identity.

This idea appeared especially attractive because it keeps the protocol surface small. It does not require modifying threshold issuance and it does not appear to interfere with Coconut's proof system. The tag is computed by the user and provided to the verifier as an additional value alongside the standard presentation.

6.1.2 Deterministic Tags Break Unlinkability

At first glance, Coconut's re-randomization mechanism appears sufficient to eliminate stable identifiers across presentations. However, the construction includes a deterministic tag derived from the exponent m, which is preserved across re-randomized presentations. This tag is explicitly designed to be validated

by the verifier in a pairing-friendly setting and therefore remains invariant under re-randomization. As a result, even if the credential itself is perfectly re-randomized, deterministic functions of m can act as stable linking handles. This breaks unlinkability at the presentation level, independently of the randomness applied elsewhere in the protocol.

Consider two verifiers with identifiers id_{v_1} and id_{v_2}. They observe tags

$$\mathrm{tag}_{v_1} = H(\mathrm{id}_{v_1})^m, \qquad \mathrm{tag}_{v_2} = H(\mathrm{id}_{v_2})^m.$$

By bilinearity, they can perform an immediate cross check:

$$e\big(H(\mathrm{id}_{v_1}), \mathrm{tag}_{v_2}\big) = e\big(H(\mathrm{id}_{v_2}), \mathrm{tag}_{v_1}\big).$$

This equality holds if and only if both tags were generated using the same exponent m. Put differently, two verifiers can now determine that two unlinkable looking presentations came from the same underlying credential, even though the credential itself was re randomized correctly.

6.1.3 What we gained and what we lost

For a single verifier, tags provide local reuse detection. Repeated presentations produce the same tag, enabling rate limiting and preventing naive replay. However, the same mechanism also allows colluding verifiers to establish cross-verifier links. In exchange for local Sybil resistance, the construction sacrifices one of Coconut's key privacy properties: unlinkability across relying parties.

This behavior does not stem from an implementation error, but from the structure of the construction. The same exponent m is used in contexts where verifiers reason about its algebraic properties. While re-randomization protects the representation of a signature, it does not prevent linkability once deterministic, pairing-checkable functions of an exponent are exposed and implicitly validated by the scheme.

6.1.4 Takeaway and future direction

This attempt clarified a design lesson that shaped the rest of our thinking:

> *In pairing based credential systems, any deterministic value derived from an algebraically validated exponent can become a cross verifier linking handle.*

As a consequence, transplanting zk-creds-style controlled linkability into Coconut requires more care than exporting a tag derived from signed attributes. A more promising direction is to enforce local linkability by construction, using secrets that are not algebraically coupled to Coconut's verification equations, while optionally proving correct formation inside the presentation proof. Although this chapter does not fully develop such a construction, the tag experiment is informative in that it makes the failure mode explicit and illustrates why naive approaches break Coconut's privacy guarantees.

6.2 Revocation Attempts: Accumulator Inspired Direction

The comparative analysis also suggested revocation as a missing operational feature in Coconut relative to some accumulator based systems. A natural first direction was to treat revocation as a non-revocation proof, where the user proves that a hidden revocation handle is a member of a public set of valid credentials or equivalently not present in a set of revoked credentials.

The immediate challenge is that revocation introduces system state that changes over time. Coconut's strongest properties are easiest to maintain when the verifier checks a stateless proof. Accumulator-based revocation can preserve anonymity and unlinkability, but it requires a careful interface: the verifier must

obtain the current accumulator value and the user must update or refresh a witness as the set evolves. In a threshold issuance setting, this also raises governance questions about who updates the revocation state, how updates are authenticated and how users learn the latest state without introducing a new tracking channel.

In this thesis we did not implement a full accumulator based revocation layer, but the exploration clarified two concrete requirements for a compatible design: the revocation handle must remain hidden during presentations and any witness update mechanism must avoid introducing a stable identifier across sessions.

6.3 Operational Cost of Updates: Re-issuance versus Incremental Changes

Finally, the comparative analysis highlighted a practical operational weakness that appears in many credential systems: a single attribute update can imply full re-issuance. In Coconut, attributes are bound into a single signature exponent structure. From an operational point of view, changing one attribute such as role, status or expiration can require a new issuance round with the authorities, including blind issuance and threshold aggregation.

We briefly explored whether ideas from other systems could reduce this overhead. The main obstacle is that Coconut's compactness comes from binding many attributes into one signature object, so incremental updates tend to re-introduce either additional signatures, additional proofs or new system state. The design space includes approaches such as separating long-lived and short-lived attributes, using layered credentials where only a subset is re-issued or introducing an external validity token that can be rotated without changing the base credential. Each option trades operational efficiency against complexity and, potentially, new trust assumptions.

While we do not present a finalized update mechanism in this thesis, the exploration motivates future work on modular credential composition, where Coconut could certify a stable core and a lighter mechanism could manage frequently changing attributes.

7 Conclusion and Future Work

7.1 Conclusion

This thesis was motivated by a simple idea: digital identity should feel like proving a condition rather than exposing a profile. Anonymous credentials support that shift by allowing holders to prove policy-relevant statements while minimizing disclosure and limiting linkability across repeated uses. When issuance is decentralized, however, the central question becomes broader than protocol design alone. What matters is which security and privacy properties are gained or weakened and how cost and complexity move across the credential lifecycle.

To address this question, the thesis makes two concrete contributions. First, it introduces a clear and consistent comparison frame for decentralized anonymous credential schemes, distinguishing properties such as trust distribution, unlinkability, selective disclosure, revocation and governance so that similar terminology across papers does not hide different assumptions. Second, it implements a reproducible benchmark that translates theoretical trade-offs into measurable patterns, enabling concrete reasoning about where computation and payload costs land in practice.

The empirical evaluation highlights that Coconut and zk-creds follow two distinct lifecycle strategies rather than forming a simple faster versus slower comparison. In the evaluated flow, Coconut places more effort in `issue` and keeps `show` relatively lightweight, which aligns with settings where credentials are presented frequently and holder-side latency is important. zk-creds shifts the dominant cost to `show` through circuit-based proving while keeping `verify` extremely fast, which aligns with verifier-heavy settings that require scalable validation. More generally, the results illustrate a key systems insight: decentralization does not remove trust, it redistributes it across committees, public state, governance choices and update mechanisms.

The benchmark is not intended to produce a universal ranking. Its purpose is to provide reproducible engineering evidence that supports deployment decisions by making cost placement explicit. For that reason, conclusions are restricted to what is directly measured: zk-creds performance is circuit-dependent and the current setup evaluates a single concrete flow, resulting in one strictly comparable point between both systems. Within this scope, the main takeaway is practical. The best construction is the one that places the dominant cost where the deployment can afford it, without sacrificing privacy guarantees or operational feasibility.

These findings also motivate the next step, namely extending both the design space and the benchmark coverage to map trade-offs more fully.

7.2 Future work

Two directions stand out.

On the design side, the exploratory line of work around Coconut can be extended toward features that often determine real-world viability. In particular, incorporating Sybil resistance, revocation and improved handling of updates and re-issuance would address known practical gaps, but doing so without weakening unlinkability is not straightforward. It requires committing to a concrete model of public state, update frequency, governance rules and carefully evaluating how these choices affect both privacy and operational cost.

On the evaluation side, the most impactful extension is to broaden the zk-creds benchmark beyond a single circuit. Because proving time, verifier payload and overall cost depend strongly on the chosen predicate, adding circuits of increasing complexity would yield a more representative map of trade-offs. This would enable more realistic conclusions about how costs evolve as policy logic becomes richer and when fast verification is worth the proving burden placed on holders.

Use of AI Tools

AI-based tools were used during the preparation of this thesis to support language editing, clarity of exposition and iterative refinement of the text. All technical content, system design decisions, experimental methodology, results, analysis and conclusions were produced and validated by the author. The use of such tools did not replace critical reasoning, original contribution or independent verification of the material.

Bibliography

[1] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2014.

[2] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2019.

[3] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 790–808. IEEE, 2023.

[4] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

[5] Alberto Sonnino and contributors. Coconut reference implementation. `https://github.com/asonnino/coconut`, 2019. GitHub repository.

[6] Michael Rosenberg and contributors. zkcreds-rs implementation. `https://github.com/rozbb/zkcreds-rs`, 2023. GitHub repository.

[7] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Topics in Cryptology – CT-RSA*, pages 111–126. Springer, 2016.

[8] Carmen González Luque. Tfm credential benchmark harness. `https://github.com/carmenzalez/tfm-credential-benchmark`, 2026. GitHub repository.

A
Benchmark source code

This appendix collects the key implementation excerpts used to instrument the benchmark described in Chapter 4 and analyzed in Chapter 5. The goal is to make the measurement boundaries and size accounting explicit without duplicating the full codebase in the thesis.

The complete implementation, together with instructions to reproduce all experiments and figures, is available in the project repository. The snippets shown here focus on the parts that define what is counted as `issue`, `show` and `verify` and on how storage size and verifier side payload are estimated for Coconut and zk-creds.

The complete implementation and reproduction instructions are available in the project repository [8].

A.1 Coconut times accounting

```python
    # ----------------------
    # ISSUE: blind issuance + threshold signing + aggregation
    # ----------------------
    t0 = time.perf_counter()
    Lambda = prepare_blind_sign(params, gamma, private_m, public_m=public_m)

    # Each of the t signers issues a blind signature share
    sigs_tilde = [
        blind_sign(params, ski, gamma, Lambda, public_m=public_m)
        for ski in signers
    ]

    # Unblind each signature share using ElGamal secret d
    sigs = [unblind(params, sigma_tilde, d) for sigma_tilde in sigs_tilde]

    # Aggregate signature shares into a final credential sigma
    sigma = agg_cred(params, sigs)
    t1 = time.perf_counter()

    # ----------------------
    # SHOW/PROVE: create a presentation/proof Theta from sigma and private attributes
    # ----------------------
    Theta = prove_cred(params, aggr_vk, sigma, private_m)
    t2 = time.perf_counter()

    # ----------------------
    # VERIFY: verifier checks Theta using public attributes
    # ----------------------
    ok = verify_cred(params, aggr_vk, Theta, public_m=public_m)
    t3 = time.perf_counter()
    assert ok is True
```

Figure A.1: Timing boundaries for Coconut.

A.2 Coconut size accounting

```python
    return {
        "issue_ms": (t1 - t0) * 1000.0,
        "show_ms": (t2 - t1) * 1000.0,
        "verify_ms": (t3 - t2) * 1000.0,

        # Object sizes
        "credential_bytes": obj_size_bytes(sigma),
        "presentation_bytes": obj_size_bytes(Theta),

        # Optional network approx.
        "public_input_bytes": obj_size_bytes(public_m),
        "network_verify_bytes": obj_size_bytes(Theta) + obj_size_bytes(public_m),
    }
```

Figure A.2: Size estimation logic for Coconut.

A.3 zk-creds times accounting

```
78          // ---------------------------
79          // ISSUE: create token + commitment
80          // ---------------------------
81          let start_issue = Instant::now();
82
83          // User computes a multishow token
84          let nonce = Fr::rand(&mut rng);
85          let ctr: u16 = 1;
86          let token = MultishowableAttrs::<_, TestComSchemePedersen>::compute_presentation_token(
87              &person,
88              params.clone(),
89              epoch,
90              ctr,
91              nonce,
92          )
93          .unwrap();
94
95          // Pedersn commitment for the attrb (stored cred)
96          let person_com = Attrs::<_, TestComSchemePedersen>::commit(&person);
97
98          // Record issuance time in ms
99          let issue_ms = start_issue.elapsed().as_secs_f64() * 1000.0;
100
```

Figure A.3: Issue time measurement in zk-creds.

```rust
    // ----------------------------
    // SHOW: prove (timed)
    // ----------------------------
    // User constructs a checker for their predicate
    let users_checker = RevealingMultishowChecker {
        token: token.clone(),
        epoch,
        nonce,
        max_num_presentations,
        ctr,
        params: params.clone(),
    };
    let start_show = Instant::now();

    // Prove the predicate
    let proof = prove_birth(&mut rng, &pk, users_checker, person.clone()).unwrap();

    let show_ms = start_show.elapsed().as_secs_f64() * 1000.0;

    // ------------------------------
    // VERIFY (timed)
    // ------------------------------
    // Now verify the predicate
    // Make the checker with only the public data
    let verifiers_checker = RevealingMultishowChecker {
        token: token.clone(),
        epoch,
        nonce,
        max_num_presentations,
        params: params.clone(),
        ..Default::default()
    };

    let start_verify = Instant::now();
    let ok = verify_birth(&vk, &proof, &verifiers_checker, &person_com).unwrap();
    let verify_ms = start_verify.elapsed().as_secs_f64() * 1000.0;
    assert!(ok);

```

Figure A.4: Show and verify time measurement in zk-creds.

A.4 zk-creds size accounting

```rust
            // ============================
            // SIZE ESTIMATES (bytes)
            // ============================
            // Token size is approximated as the serialized sizes of its core components
            let token_bytes = ser_len(&token.hidden_ctr) + ser_len(&token.hidden_line_point);

            // Commitment size is the serialized size of the Pedersen commitment to the attributes.
            let commitment_bytes = ser_len(&person_com);

            // Proof size is the serialized size of the ZK proof object.
            let proof_bytes = ser_len(&proof);

            // In this model, the credential is just the commitment.
            let credential_bytes = commitment_bytes;

            // Public inputs required by the verifier.
            let public_inputs: Vec<Fr> = vec![
                Fr::from(epoch),
                nonce,
                token.hidden_ctr,
                token.hidden_line_point,
                Fr::from(max_num_presentations),
            ];

            let public_input_bytes: usize = {
                let mut v = Vec::new();
                public_inputs.serialize(&mut v).unwrap();
                v.len()
            };

            // Approximate protocol metadata bytes to count as "sent over the network".
            // epoch(u64)=8, ctr(u16)=2, max(u16)=2, nonce (~32 bytes).
            let meta_bytes = 8 + 2 + 2 + 32;

            // Presentation payload size: proof + token + commitment + metadata.
            // (This is what the prover would send in a "show" message in this model)
            let presentation_bytes = proof_bytes + token_bytes + commitment_bytes + meta_bytes;

            // Total bytes needed for verification in this model: proof + public inputs + commitment + metadata.
            // NOTE: token_bytes is not included here because the public inputs represent
            // the verifier-facing values extracted from token/params.
            let network_verify_bytes = proof_bytes + public_input_bytes + commitment_bytes + meta_bytes;

```

Figure A.5: Size estimation logic for zk-creds.

B
Benchmark Tables

This appendix provides supplementary tables corresponding to the runtime measurements. The tables are included to make small timing values, particularly in the verification stage, explicit and easier to inspect than in the aggregated plots.

Each table shows a subset of the raw per-run measurements produced by the benchmark pipeline for a specific operation (`issue`, `show`, or `verify`) and system (Coconut or zk-creds). For readability, the tables displayed here contain 10 representative measurements extracted from the full dataset. In all cases, the reported median values and figures in Chapter 5 are computed over 100 executions per operation, not only the samples shown in these tables.

The complete set of raw measurements is preserved in the benchmark outputs and was used for all statistical summaries and plots discussed in the evaluation.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
10400	zkcreds	revealing_multishow_birth	issue	2	0	0	0	0	0.6104	None
10411	zkcreds	revealing_multishow_birth	issue	2	0	0	0	1	0.6531	None
10422	zkcreds	revealing_multishow_birth	issue	2	0	0	0	2	0.5996	None
10433	zkcreds	revealing_multishow_birth	issue	2	0	0	0	3	0.5855	None
10444	zkcreds	revealing_multishow_birth	issue	2	0	0	0	4	0.5836	None
10455	zkcreds	revealing_multishow_birth	issue	2	0	0	0	5	0.5868	None
10466	zkcreds	revealing_multishow_birth	issue	2	0	0	0	6	0.5776	None
10477	zkcreds	revealing_multishow_birth	issue	2	0	0	0	7	0.5785	None
10488	zkcreds	revealing_multishow_birth	issue	2	0	0	0	8	0.5862	None
10499	zkcreds	revealing_multishow_birth	issue	2	0	0	0	9	0.5862	None

Figure B.1: Sample runtime measurements for the `issue` operation in zk-creds.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
0	coconut	show_verify	issue	2	0	3	4	0	47.3568	None
8	coconut	show_verify	issue	2	0	3	4	1	47.6262	None
16	coconut	show_verify	issue	2	0	3	4	2	52.9579	None
24	coconut	show_verify	issue	2	0	3	4	3	46.8535	None
32	coconut	show_verify	issue	2	0	3	4	4	48.0421	None
40	coconut	show_verify	issue	2	0	3	4	5	48.0279	None
48	coconut	show_verify	issue	2	0	3	4	6	47.4503	None
56	coconut	show_verify	issue	2	0	3	4	7	63.704	None
64	coconut	show_verify	issue	2	0	3	4	8	48.4279	None
72	coconut	show_verify	issue	2	0	3	4	9	48.0595	None

Figure B.2: Sample runtime measurements for the `issue` operation in Coconut.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
10401	zkcreds	revealing_multishow_birth	show	2	0	0	0	0	599.0675	None
10412	zkcreds	revealing_multishow_birth	show	2	0	0	0	1	657.5159	None
10423	zkcreds	revealing_multishow_birth	show	2	0	0	0	2	905.3611	None
10434	zkcreds	revealing_multishow_birth	show	2	0	0	0	3	635.9705	None
10445	zkcreds	revealing_multishow_birth	show	2	0	0	0	4	613.5594	None
10456	zkcreds	revealing_multishow_birth	show	2	0	0	0	5	672.4702	None
10467	zkcreds	revealing_multishow_birth	show	2	0	0	0	6	612.5699	None
10478	zkcreds	revealing_multishow_birth	show	2	0	0	0	7	593.1092	None
10489	zkcreds	revealing_multishow_birth	show	2	0	0	0	8	600.6697	None
10500	zkcreds	revealing_multishow_birth	show	2	0	0	0	9	596.2931	None

Figure B.3: Sample runtime measurements for the `show` operation in zk-creds.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
1	coconut	show_verify	show	2	0	3	4	0	13.9268	None
9	coconut	show_verify	show	2	0	3	4	1	13.8099	None
17	coconut	show_verify	show	2	0	3	4	2	13.8509	None
25	coconut	show_verify	show	2	0	3	4	3	13.9969	None
33	coconut	show_verify	show	2	0	3	4	4	14.498	None
41	coconut	show_verify	show	2	0	3	4	5	14.0015	None
49	coconut	show_verify	show	2	0	3	4	6	14.0971	None
57	coconut	show_verify	show	2	0	3	4	7	15.1926	None
65	coconut	show_verify	show	2	0	3	4	8	13.794	None
73	coconut	show_verify	show	2	0	3	4	9	14.0736	None

Figure B.4: Sample runtime measurements for the `show` operation in Coconut.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
10402	zkcreds	revealing_multishow_birth	verify	2	0	0	0	0	3.5482	None
10413	zkcreds	revealing_multishow_birth	verify	2	0	0	0	1	4.4902	None
10424	zkcreds	revealing_multishow_birth	verify	2	0	0	0	2	3.6636	None
10435	zkcreds	revealing_multishow_birth	verify	2	0	0	0	3	3.7895	None
10446	zkcreds	revealing_multishow_birth	verify	2	0	0	0	4	3.5143	None
10457	zkcreds	revealing_multishow_birth	verify	2	0	0	0	5	3.5506	None
10468	zkcreds	revealing_multishow_birth	verify	2	0	0	0	6	3.4893	None
10479	zkcreds	revealing_multishow_birth	verify	2	0	0	0	7	3.6133	None
10490	zkcreds	revealing_multishow_birth	verify	2	0	0	0	8	3.4985	None
10501	zkcreds	revealing_multishow_birth	verify	2	0	0	0	9	3.5608	None

Figure B.5: Sample runtime measurements for the `verify` operation in zk-creds.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
2	coconut	show_verify	verify	2	0	3	4	0	35.0113	None
10	coconut	show_verify	verify	2	0	3	4	1	30.8271	None
18	coconut	show_verify	verify	2	0	3	4	2	30.883	None
26	coconut	show_verify	verify	2	0	3	4	3	30.7159	None
34	coconut	show_verify	verify	2	0	3	4	4	30.8425	None
42	coconut	show_verify	verify	2	0	3	4	5	30.282	None
50	coconut	show_verify	verify	2	0	3	4	6	31.2439	None
58	coconut	show_verify	verify	2	0	3	4	7	31.584	None
66	coconut	show_verify	verify	2	0	3	4	8	30.0499	None
74	coconut	show_verify	verify	2	0	3	4	9	32.2602	None

Figure B.6: Sample runtime measurements for the `verify` operation in Coconut.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
4	coconut	show_verify	credential_size	2	0	3	4	0	None	66
12	coconut	show_verify	credential_size	2	0	3	4	1	None	66
20	coconut	show_verify	credential_size	2	0	3	4	2	None	66
28	coconut	show_verify	credential_size	2	0	3	4	3	None	66
36	coconut	show_verify	credential_size	2	0	3	4	4	None	66
44	coconut	show_verify	credential_size	2	0	3	4	5	None	66
52	coconut	show_verify	credential_size	2	0	3	4	6	None	66
60	coconut	show_verify	credential_size	2	0	3	4	7	None	66
68	coconut	show_verify	credential_size	2	0	3	4	8	None	66
76	coconut	show_verify	credential_size	2	0	3	4	9	None	66

Figure B.7: Sample stored credential sizes for Coconut.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
10404	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	0	None	32
10415	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	1	None	32
10426	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	2	None	32
10437	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	3	None	32
10448	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	4	None	32
10459	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	5	None	32
10470	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	6	None	32
10481	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	7	None	32
10492	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	8	None	32
10503	zkcreds	revealing_multishow_birth	credential_size	2	0	0	0	9	None	32

Figure B.8: Sample stored credential sizes for zk-creds.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
7	coconut	show_verify	network_verify_size	2	0	3	4	0	None	355
15	coconut	show_verify	network_verify_size	2	0	3	4	1	None	355
23	coconut	show_verify	network_verify_size	2	0	3	4	2	None	355
31	coconut	show_verify	network_verify_size	2	0	3	4	3	None	355
39	coconut	show_verify	network_verify_size	2	0	3	4	4	None	355
47	coconut	show_verify	network_verify_size	2	0	3	4	5	None	355
55	coconut	show_verify	network_verify_size	2	0	3	4	6	None	355
63	coconut	show_verify	network_verify_size	2	0	3	4	7	None	355
71	coconut	show_verify	network_verify_size	2	0	3	4	8	None	355
79	coconut	show_verify	network_verify_size	2	0	3	4	9	None	355

Figure B.9: Sample verifier-facing payload sizes for Coconut.

Medians (bars) Table

	scheme	flow	operation	n_attrs_total	n_revealed	t	n	run_id	time_ms	size_bytes
10407	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	0	None	436
10418	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	1	None	436
10429	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	2	None	436
10440	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	3	None	436
10451	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	4	None	436
10462	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	5	None	436
10473	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	6	None	436
10484	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	7	None	436
10495	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	8	None	436
10506	zkcreds	revealing_multishow_birth	network_verify_size	2	0	0	0	9	None	436

Figure B.10: Sample verifier-facing payload sizes for zk-creds.

C

Coconut Issuance (Non-Threshold)

C.1 Coconut (Non-Threshold) Signature on a Single Attribute

C.1.1 Coconut (Non-Threshold) Signature on a Single Attribute

Coconut issuance flow (without threshold) for a single private attribute m. The scheme is based on Pointcheval–Sanders signatures.

C.1.1.1 Authority Key Generation

The authority samples two secret exponents

$$(x, y)$$

and publishes the corresponding public key elements

$$(g^x,\, g^y)$$

where g is a generator of the group G_1. The user computes a commitment to his attribute:

$$C_p = g^t \cdot g^{y \cdot m}.$$

C.1.1.2 Blind Issuance by the Authority

The authority samples a random exponent

$$u \xleftarrow{\$} \mathbb{Z}_q.$$

The blind signature is

$$\widetilde{s} = (g^x \cdot C_p)^u.$$

The authority returns to the user

$$(h, \widetilde{s}),$$

where

$$h = g^u.$$

C.1.1.3 Unblinding

The user needs to unblind the signature to use it:

$$s = \widetilde{s} \cdot (h^{-t}).$$

Substituting the expression for $\widetilde{s}$ and h:

$$\begin{aligned} s &= (g^x \cdot g^{t+y\cdot m})^u \cdot g^{-u\cdot t} \\ &= g^{u\cdot x} \cdot g^{u(t+y\cdot m)} \cdot g^{-u\cdot t} \\ &= g^{u\cdot x} \cdot g^{u\cdot y\cdot m} \\ &= g^{(x+y\cdot m)\cdot u}. \end{aligned}$$

We have removed t that was the hiding factor:

$$\boxed{s = g^{(x+y\cdot m)\cdot u}}.$$

C.1.2 Coconut (Non-Threshold) Signature on a Pair of Attributes

- a**:** one attribute
- w**:** whitecard attribute for linkage

C.1.2.1 Background: Multi-attribute Coconut paper

The theoretical definition of multi-attribute credentials in Coconut is explained in:

- **Section III, Subsection E — Multi-Attribute Credentials**

There, the paper states:

> *We expand our scheme to embed multiple attributes into a single credential without increasing its size. The authorities' key pairs become*

$$\begin{aligned} sk &= (x, y_1, \ldots, y_q) \\ vk &= (g_2^x,\, g_2^{y_1},\, \ldots,\, g_2^{y_q}) \end{aligned}$$

where q is the number of attributes.

This means:

- each attribute has an associated **secret coefficient** y_i,

and the credential signs the vector of attributes

$$(m_1,\, m_2,\, \ldots,\, m_q).$$

C.1.2.2 Authority Key Generation

$$C_p = g^t \cdot g^{y_a\cdot a} \cdot g^{y_w\cdot w}.$$

C.1.2.3 Blind Issuance by the Authority

The authority samples a random exponent

$$u \xleftarrow{\$} \mathbb{Z}_q.$$

The blind signature is

$$\widetilde{s} = (g^x \cdot C_p)^u.$$

The authority returns to the user

$$(h, \widetilde{s}),$$

where

$$h = g^u.$$

C.1.2.4 Unblinding

The user needs to unblind the signature to use it:

$$s = \widetilde{s} \cdot (h^{-t}).$$

Substituting the expression for $\widetilde{s}$ and h:

$$\begin{aligned} s &= (g^x \cdot g^{t+y_a \cdot a + y_w \cdot w})^u \cdot g^{-u \cdot t} \\ &= g^{u \cdot x} \cdot g^{u(t+y_a \cdot a + y_w \cdot w)} \cdot g^{-u \cdot t} \\ &= g^{u \cdot x} \cdot g^{u \cdot t} \cdot g^{u \cdot y_a \cdot a} \cdot g^{u \cdot y_w \cdot w} \cdot g^{-u \cdot t} \\ &= g^{(x + y_a \cdot a + y_w \cdot w) \cdot u}. \end{aligned}$$

We have removed t that was the hiding factor:

$$\boxed{s = g^{(x + y_a \cdot a + y_w \cdot w) \cdot u}}.$$