3PBCS

A Privacy-Preserving, Personhood-Based Credential System

by **Ksandros Apostoli**

Approved by the Examining Committee:

Prof. Dr. Bryan Ford
Thesis Advisor

Simone Colombo
Thesis Supervisor

A thesis presented for the degree of
Master of Science

Decentralized and Distributed Systems Laboratory
Department of Computer Science (IN-S)
École Polytechnique Fédérale de Lausanne
Switzerland
March 4, 2022

Ksandros Apostoli*, Simone Colombo, and Bryan Ford

A Privacy-Preserving, Personhood-Based Credential System

Abstract: Recent anonymous credential schemes present major advances in the way digital identities are treated, empowering users with fine-grained control over their personal data. While limiting the amount of user-data that is accessible to third parties paves the way to stronger privacy guarantees, it also leads to new challenges in preserving accountability and Sybil-Resistance. In this work, we present **3PBCS**, a novel credential system, which integrates existing anonymous credential schemes with the notion of Proof-of-Personhood to achieve privacy, accountability and Sybil-Protection. Our scheme relies on Proof-of-Personhood Tokens for bootstrapping digital identities. A combination of classic MixNets and Secure Multiparty Computation is leveraged to achieve Sybil-Resistance as well as accountability in a fully privacy-preserving manner. Among usability features, we put emphasis on the support for multiple pseudonymous accounts for a single person, without sacrificing any of the above-mentioned guarantees. These characteristics, make our proposed work suitable for use in a wide variety of applications, including social networks, where privacy, accountability and Sybil-protection are becoming crucial, and where support for multiple accounts is a core requirement.

Keywords: Proof-of-Personhood, Anonymous Credentials, Sybil-Resistance, Privacy, Accountability

1 Introduction

Humans today heavily rely on digital services for completing daily tasks [1, 2], and the majority of these services base their access-granting mechanisms on user identification [3]. This has led to an increase in the importance of digital identities and credential systems used to manage them. Moreover, the widespread use of credentials in today's digital world, has come with several profound challenges that credential systems need to address. Some of these challenges are reflected in a series of rising concerns such as:

1. ***Privacy***. The mass adoption of the Internet for personal and professional use over the last decades has been accompanied with an ever-growing concern over users' privacy [4]. This presents the first major challenge in the design and use of digital identities. Privacy is primarily affected by the quantity and type of user-related data that is used in the creation and utilisation of credentials, as well as the location where these data reside.
2. ***Sybil-Attacks***. Digital information and metrics have taken a central role in our societies. Artists are now being valued based on monthly listeners, e-commerce websites build their trust based on user reviews, while studies have shown an increased association between likes and self-worth among individuals [5]. Therefore, securing the legitimacy of such metrics is crucial. Malicious users capable of creating infinite digital identities present a major concern under this context, as they can misuse this capability to generate an artificial influence over the digital environment. This problem, known as a *Sybil Attack* [6], poses the second challenge that any contemporary credential system needs to consider.
3. ***Lack of Accountability***. Apart from the risk of fictive digital feedback of quantitative nature, the qualitative nature of user-generated information plays an equally important role towards a safe digital realm. To illustrate this, consider the example of fake news on digital platforms, which in extreme occasions, have proven to be a key contributor to opinion influenced politics [7]. While manual content reviews constitute a first mitigating step against misinformation, proper accountability measures are necessary to effectively prevent such cases. Proper blacklisting should make it impossible for malefactors to misbehave again. *Accountability* therefore, is

***Corresponding Author: Ksandros Apostoli:** École Polytechnique Fédérale de Lausanne, E-mail: ksandros.apostoli@epfl.ch
Simone Colombo: École Polytechnique Fédérale de Lausanne, E-mail: simone.colombo@epfl.ch
Bryan Ford: École Polytechnique Fédérale de Lausanne, E-mail: bryan.ford@epfl.ch

a must for any credential system, posing the third major challenge.

When it comes to *privacy*, recent advances on anonymous credential schemes have demonstrated great results. Schemes like *Coconut* [8], aim at applying the principles of "*Least Privilege*" and "*Minimum Exposure*" on credential attributes, handing back to users fine-grained control and full ownership over their personal data. Despite enabling state-of-the-art privacy, through limiting the amount of information provided to verifiers, like in the case of *Coconut*, no anonymous credential scheme to our knowledge encompasses Accountability and Sybil-Resistance in their design goals.

One approach for enabling Sybil-Resistance and Accountability in the use of digital identities, relies on binding these identities to physical existence, i.e. *personhood*. This is known as *Proof-of-Personhood* [9]. Classic proposals in the usage of Proof-of-Personhood, involve the employment of *Linkable Ring Signatures* [10]. Linkable Ring Signatures represent an anonymous signature primitive, that enables the generation of unique, verifiable linkage tags per signer. These tags can be used as a strong *Sybil-Protection* and *accountability* mechanism.

However, unlike anonymous credential schemes, the privacy guarantees in the use of Proof-of-Personhood combined with Linkable Ring Signatures are reduced to a minimum. The anonymity features of Linkable Ring Signatures hide the true identity of the user, but fail to protect against intricate privacy issues, such as activity tracking. Moreover, treated as a simple a public key, Proof-of-Personhood suffers from basic utilisation barriers, such as support for multiple pseudonymous accounts belonging to the same person. The lack of such functional requirements presents considerable usability limitations. This is particularly true in the case of social platforms, where users should be able to project different aspects of their lives to different accounts.

Goals and Contribution. This work aims to leverage the complementary features offered by the two approaches described above, combining them into a single personhood-based credential system. Concretely, we base our proposal on the *Coconut* [8] credential scheme, as well as Proof-of-Personhood tokens [11] obtained through classic Pseudonymous Party set-ups. The 3PB Credential Scheme aims at covering the above-described guarantees through the following mechanisms:

- ***Sybil-Resistance***. Each issued credential is bound to a secret key representing the user's personhood. This key, is used to generate context-specific linkage tags for each Sybil-sensitive action within end-services. Being unique *per person*, these tags provide strong Sybil-resistance in even in the case of multiple pseudonyms being created by the same person. Being unique *per action*, avoids the misuse of these tags for activity tracking.
- ***Accountability***. 3PBCS enforces blacklisting in a privacy-preserving manner. Similar to Sybil-Resistance, this is achieved on a personhood level, with context-specific linkage tags serving as the entries of blacklists. A new blacklist is computed for each context dynamically in a secure multiparty computation manner, to preserve the privacy goals described below.
- ***Privacy***. Core privacy guarantees of anonymous credential systems are inherited from the *Coconut* Credential Scheme. The main privacy concerns in our work arise when attempting to enforce *Sybil-Protection* and *Accountability*:
  1. *Tracking Prevention.* The context-specific nature of linkage tags prevent a single service, or multiple colluding services from misusing these tags to track a user across different contexts. Similarly, 3PBCS maintains blacklists that are unique per-context. To prevent tracking of user activity, updating blacklists from one context to the other is only possible to be carried in a secure multi-party computation manner.
  2. *Linking Multiple Pseudonymous Accounts.* Selective disclosure enables the separation of user pseudonyms and linkage tags, in two different, unlinkable credential presentations. The first contains the pseudonym and desired action, and is sent directly to the end-service. The second credential is completely anonymized, and only proves proper computation of the linkage tag for the desired action in zero-knowledge manner. This second instance of the credential is then transmitted together with the linkage tag through a Mix-Network for batching and shuffling before reaching the end-server. Mixing and batching ensures that any correlation between the linkage tag and the pseudonym contained in the first credential is lost to the end-service. Hence, the uniqueness property of linkage tags cannot be misused by the end-service to link multiple credentials to the same owner. Note here that votes, likes, follows and other actions leading to quantitative metrics, must be only computed by the end-service using the linkage tags received.

We start by listing a set of necessary cryptographic preliminaries. Then, we proceed with a description of the main building blocks of 3PBCS, namely the *Coconut* credential scheme, as well as the notion of *Proof-of-Personhood* (PoP) and PoP Tokens. Next, we present the detailed construction of 3PBCS, starting from personhood-based credential issuance, to the use of the credentials through our Identity and Accountability Management Authority.

Lastly, we provide an evaluation for both feasibility and performance of our scheme through the implementation of a proof-of-concept for 3PBCS. Details on the implementation as well as the preliminary performance evaluations are presented after the definition of our scheme.

2 Background and Preliminaries

First, we provide the necessary background and cryptographic building blocks composing our system. Next, we give a brief description of the *Coconut* credential scheme [8] and its privacy-preserving features. Finally we move to the concepts of Proof-of-Personhood and Pseudonymous events, highlighting how these can be leveraged for accountability and Sybil-resistance guarantees.

2.1 Cryptographic Preliminaries

A high-level presentation of the cryptographic background relevant to this work is presented below.

2.1.1 Zero-Knowledge Proofs

Zero knowledge proofs are cryptographic protocols that allow a prover P to prove to a verifier V, that they have knowledge over certain statements, without revealing any additional information other than the validity of the statements themselves. For instance, a user might want to prove to a third-party, that they know the secret pre-image s for some hash $h = H(s)$, without explicitly revealing the value of s.

Throughout this paper, we employ the standard notation for zero-knowledge proof schemes from Camenisch-Stadler [12]:

$$ZKP_\phi \{(x) : \phi(x, y)\}$$

where ϕ denotes the "statement" or predicate that is to be proven, consisting on some private knowledge x and some public knowledge y. Then, upon receiving this zero-knowledge proof from prover P, the verifier V, can verify that the statement $\phi(x, y)$ holds for x and y.

A more formal definition of Zero-Knowledge proof-of-knowledge schemes follows:

Definition 1 (Zero-Knowledge Proof scheme). Let P denote a prover and V denote a verifier. Moreover, let $\mathcal{L}$ denote a language, i.e. a set of statements together with knowledge about these statements.

Then, a Zero-Knowledge Proof (ZKP) scheme, satisfies the following properties:

1. (*Completeness*) If a predicate $\phi(x, y) \in \mathcal{L}$ is true, a honest (one properly following the protocol) verifier V will be convinced of this fact by interacting with a honest prover P.
2. (*Soundness*) If a predicate $\phi(x, y) \in \mathcal{L}$ is false, a cheating prover P can convince a honest verifier V only with negligible probability.
3. (*Zero-Knowledge*) If a predicate $\phi(x, y) \in \mathcal{L}$ is true, no verifier V can learn anything about the statement other than the fact that the statement is true. In other terms, no verifier V can learn anything about the secret knowledge x in $\phi(x, y)$.

Non-Interactive Zero-Knowledge proofs ($NIZK$) extend the definition above, by additionally requiring that no interaction takes place between the prover P and verifier V.

2.1.2 Secure Multi-Party Computation

Secure multiparty compuation (SMPC) [13] describes the joint computation of a function that depends on private inputs from a group of independent data owners, who do not trust each other, or any common third party.

2.1.3 Linkable Ring Signatures

A ring signature [14] is an anonymous signature that can be spontaneously generated by any member out of a group of users. The public keys of the group members form the anonymity set. Among security guarantees, ring signatures ensure that it is infeasible to determine which of the keys from the anonymity set was used to generate a given signature.

Linkable ring signatures [10] extend classic ring signature schemes with the notion of linkability, by allowing a verifier to link two signatures that were generated by the same member without revealing the member's identity. We follow up with a formal description of the construct.

Definition 2 (Linkable Ring Signature). Let $\mathcal{U}$ be the set of r users, each associated with a public key pk_u of a standard signature scheme, where $(pk_u, sk_u) \in \mathcal{R}$, such that $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ denotes a secret-public key relation. We call $\mathcal{U}$ the *ring*. Let $\mathcal{L} = \{pk_1, \ldots, pk_r\}$. Then, let the s-th member be the signer and denote their public key as $pk_s \in \mathcal{L}$ and the corresponding secret key sk_s. The generic Linkable Ring Signature Scheme is then described by the following:

- ◊ **LinkableRing.Sign**$(m, \mathcal{L}, sk_s) \to \sigma, L$:
  Output
  $$L = H(\mathcal{L})^{sk_s}$$
  and
  $$\sigma = SPK\left\{sk_s : \vee_{i=1}^{r}\left((sk_s, pk_i) \in \mathcal{R}\right) \wedge L = H(\mathcal{L})^{sk_s}\right\}(m)$$
  where SPK denotes a *Signature based on Proof-of-Knowledge* [15].
- ◊ **LinkableRing.Verify**$(m, \sigma, \mathcal{L}) \to True/False$:
  Output $True$ if the corresponding Proof-of-Knowledge included in σ is verified to be correct. Else, output $False$.
- ◊ **LinkableRing.Link**$(L_1, L_2) \longrightarrow True/False$:
  Output $True$ if $L_1 = L_2$, $False$ otherwise.

2.1.4 Mix Networks

The term *Mix Networks* [16] or *Mix-Nets* is used to describe a cryptographic protocol, that relies on a decentralized setup for enabling communications that provide anonymity to a group of senders. This is achieved through a chain of servers known as *mixes* or *mix-servers*, that upon receiving messages from numerous senders, shuffle them, and then forward them to the next mix-server. Eventually, the set of messages reaches an *exit* mix-server node, that forwards the messages to their original destinations.

The mixing of messages that takes place on each mix-node, breaks the link between the sender of the message and the destination. Thus, tracing of end-to-end communication becomes hard for an eavesdropper. Mixnets, are based on the *anytrust* adversarial model, which includes malicious mix-nodes too, where each node only learns the previous and next hop for the set

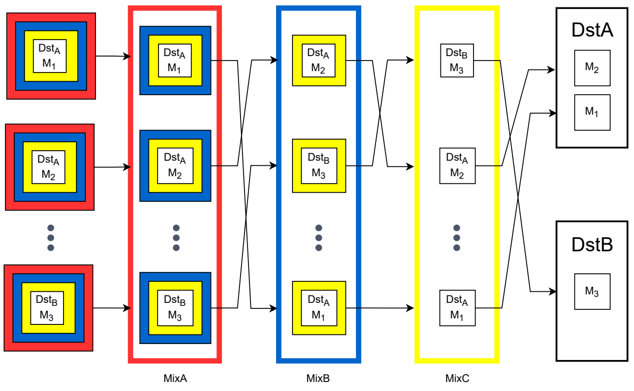

Fig. 1. Overview of Mixnets: Each mix-node removes one layer of encryption, revealing the next hop, and shuffles the order of messages before forwarding to the next server.

of messages it is expected to shuffle, but does not have visibility over the full path. This is achieved through the notion of layered decryption, where public key encryption is used by the sender to encrypt the address and optionally the contents of the message with the public keys for each of the mix-servers in the path, in the same order as the packet will reach them. Then, each mix-node removes one layer on encryption before sending the message to the next hop. Finally, the exit node removes the last layer of encryption, obtaining the address of the original destination in clear together with the message to be forwarded.

2.2 The Coconut Credential Scheme

Coconut [8] is a selective disclosure credential system, that supports distributed threshold issuance, public and private attributes, as well as re-randomization. Any user who wishes to obtain a Coconut credential, starts by preparing a request that describes a set of public or encrypted private attributes to be embedded into the credential (❶), which is then issued (❷) to a set signing authorities. Upon verifying the validity of private attributes using Zero-Knowledge Proofs-of-Knowledge, each authority responds by delivering to the user a partial credential (❸), based on a hybrid extension of Boneh–Lynn–Shacham [17] and Pointcheval-Sanders [18] signatures.

Once the user has received a threshold number of such partial credentials, they are ready to aggregate (❹) them into a single full credential, and re-randomize it (❻). When it comes to using the credential for authentication purposes, it is important to note that only users with knowledge of the private attributes contained

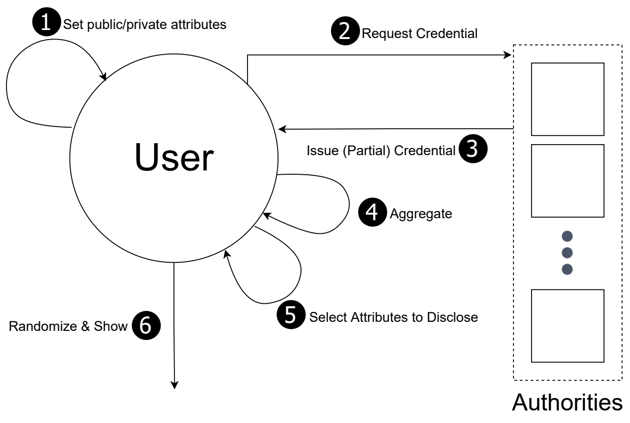
❶ Set public/private attributes
❷ Request Credential
Issue (Partial) Credential ❸
❹ Aggregate
❺ Select Attributes to Disclose
Randomize & Show ❻
User
Authorities

Fig. 2. Coconut Overview [8].

in the credential are able to do so. Moreover, any time the user decides to show the credential, they can selectively disclose attributes or statements about them (❺).

The features presented above, enable very strong privacy guarantees for users. To begin with, the user is the true owner of their data, as the authorities do not store any of the attributes. In fact some of these attributes are not even publicly shown to the authorities when the user decides to treat them as private. Moreover, not only the user can choose which attributes to disclose selectively every time they present the credential to verifiers, but they can even prove predicates or statements about certain attributes in a zero-knowledge fashion without ever disclosing them. Re-randomization provides unlinkability, in the use of the credential, another privacy advantage.

Despite the state-of-the-art privacy guarantees described above, the design goals of *Coconut* provide neither accountability nor Sybil-Resistance guarantees. Nevertheless, as argued earlier, both accountability and Sybil-Protection are at the cornerstone of any credential system today, despite being challenging to achieve in a privacy-preserving manner. 3PBCS builds on top of Coconut by adding these features while holding Coconut's privacy guarantees intact.

2.3 Proof-of-Personhood

The notion of Proof-of-Personhood (PoP)[9] describes a mechanism for binding virtual and physical identities of users while preserving users' anonymity. Such a mechanism can be used as a defence against Sybil attacks. More generally it can be utilised to enforce different accountability policies such as blacklisting, without the risk of circumvention risk of circumvention e.g., through the malicious creation of new virtual identities. From this perspective, Proof-of-Personhood can be seen as a tool for creating accountable pseudonyms.

Proof-of-Personhood relies on four main goals: *Inclusion*, *Equality*, *Security* and *Privacy*. There are numerous approaches to "obtaining" proof-of-personhood. However preliminary research has suggested that holding periodic real-world gatherings (e.g. parties) where digital and physical identities are linked, is the only method which achieves all four goals listed above. These events, known as *Pseudonym Parties* or *PoP Parties*, have been extensively studied and evaluated by [11], and will serve as the bootstrapping seeds of our credential system.

2.3.1 Pseudonym Parties

In brief, a pseudonym party [11] gives each attendee at an in-person event exactly one anonymous digital proof-of-personhood token or PoP token. The process is organized so as to leverage physical security, and the fact that real people have only one body each, to ensure that each person gets only one token. After each event, the organizers publish a list of the anonymous tokens they handed out.

A pseudonym party thus allows attendees to prove their personhood transparently in a public ceremony, demonstrating their existence as a human and obtaining a limited-term digital proof of their unique personhood, without having to be identified at all, by anyone.

2.3.2 PoP Tokens

Following the Pseudonym Party setup described above, classic proposals for the design of PoP Tokens rely on public and private key-pairs. A user u, is asked to generate a key-pair (sk_u, pk_u) prior to attending the PoP Party (❶). During the PoP Party, where the physical identity of the user is verified in an anonymous manner, the user is requested to submit the public share of their key, pk_u (❷). It is important for organizers to ensure that each user submits exactly one public key. The list of anonymous tokens published by the organizers as described above then, is simply a list of all public keys gathered from users during the event (❸).

If a user wants to prove their personhood to any third party then, they can generate a linkable

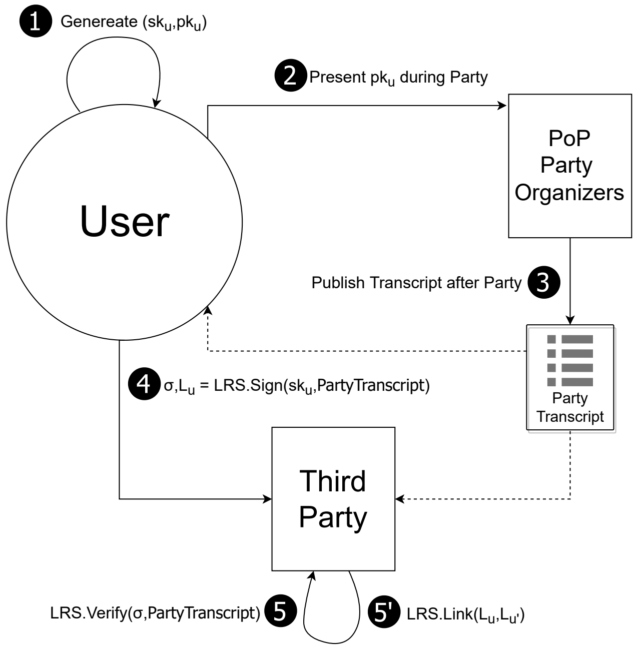
1 Genereate (sk_u,pk_u)
2 Present pk_u during Party
User
PoP Party Organizers
Publish Transcript after Party 3
Party Transcript
4 σ,L_u = LRS.Sign(sk_u,PartyTranscript)
Third Party
LRS.Verify(σ,PartyTranscript) 5
5' LRS.Link(L_u,L_u')

Fig. 3. Using PoP Tokens with Linkable Ring Signatures.

ring signature using the secret key sk_u, corresponding to their PoP Token, together with the party transcript $\mathcal{L}_{Party}$ (❹). After executing $\sigma, L =$ **LinkableRing.Sign**$(sk_u, \mathcal{L}_{Party})$, the user sends this signature σ together with the linkage tag L to the third party. The third party, can verify the signature and the validity of the linkage tag, using the procedure **LinkableRing.Verify**$(\sigma, \mathcal{L}_{Party})$ (❺). Finally, based on the uniqueness of the linkage tags across signatures, the third party can easily use them to enforce accountability and/or Sybil-Protection (❺').

Despite, enforcing strong accountability and Sybil-resistance guarantees in this scheme, the uniqueness of linkage tags can be misused by third-party services to track user activity, negatively impacting privacy. Moreover, the uniqueness of linkage tags makes the creation of multiple pseudonymous accounts impossible by once again enabling a malicious third party service to link multiple pseudonymous accounts to the same person.

3 Overview of 3PBCS

In this section we present a sketch for the architecture and system model of 3PBCS. Moreover, we describe the threat model and adversarial assumptions considered in this design.

3.1 System Actors

The 3PB Credential System is composed of three types of actors: users, a set of servers composing the credential Issuing and Accountability Management Cothority (IAMC), as well as credential verifiers often referenced as third-party services.

The *user* actor represents a subject who wishes to obtain a credential from the IAMC based on a set of claims they hold, and later use the obtained credential in order to engage with a third-party service or verifier.

Secondly, the *IAMC* actor holds three main functions. First, it carries the functionality of a *Coconut* issuing authority for providing users with credentials. Moreover, when an issued credential is to be used in a *Sybil-sensitive* scenario, the IAMC serves as a mix network for transmitting the linkage tags between the user and verifier in an anonymous manner. Finally, the IAMC nodes maintain a distributed blacklist for the verifier, which is updated dynamically as new contexts of usage are being registered.

Lastly, the *verifier* actor sits within a third-party service, and acts as a *Coconut* verifier for the credentials that are presented by users who wish to use the service. Additionally, the verifier, enforces sybil-protection, by processing the linkage tags provided by the IAMC.

3.2 Threat Model and Assumptions

In 3PBCS, users do *not* need to rely on the trustworthiness of a single server (including the IAMC nodes and third-party service). In fact, the client only needs to assume that *at least one* server is honest, without explicitly being able to identify this server. An IAMC node is *honest* if the node proceeds in its computations according to the specification of the 3PBCS protocol, and does not collude or exchange any information that is not described in the 3PBCS specification with other nodes, or the third party service. Otherwise, a node is considered *malicious*.

A malicious client may attempt to bypass Sybil-resistance measures, e.g. attempt to carry out an action more than once, when this forbidden, such as the case of voting, verifiable likes and more. Moreover, a client may wish to bypass the blacklist filtering enforced by the IAMC, i.e. attempt to carry any action $\mathcal{A}$ within the service after the user has been blacklisted due to misbehaviour reports.

A *malicious* server (including IAMC nodes and third-party service), may wish to break a user's privacy by either tracking the user's activity across different contexts within the third-party service, or by linking multiple pseudonymous accounts belonging the same person/user. A *malicious* third-party service, may even collude with other *malicious* third-party services using 3PBCS, or IAMC nodes, to fulfil these malicious goals. A *malicious* server (including IAMC nodes and third-party services) may attempt to deny service to the user, but this shall not enable it to achieve the above-listed adversarial goals.

In this work, we assume that all cryptographic primitives referenced are properly implemented and utilised. Moreover, we assume that PoP Parties have been carried according to the protocol described in [9], binding to all security guarantees provided, and that the PoP Party transcript is made publicly available by organizers. Similarly, we build on top of *Coconut* assumptions, and assume that these hold in our *Coconut* setup phase, and that verification-keys from the IAMC, corresponding to those used for credential issuance are publicly available to any third-party service wishing to use 3PBCS.

3.3 Security Goals

Our scheme focuses on three main security goals:

1. **Privacy:** Firstly, 3PBCS aims to preserve all privacy goals provided by the *Coconut* credential scheme. This is easily achieved thanks to building on top of this scheme. The main privacy challenges in 3PBCS concern *unlinkability*, and arise when attempting to integrate *Sybil-Resistance* and *accountability* to the *Coconut* scheme:
   A. *Unlinkability in user actions*, i.e. tracking prevention, where a malicious server, or numerous colluding servers attempt to link activities of the same user within the third-party service using the linkage tags used for Sybil-resistance and accountability. As described earlier, this goal is enabled by the use of context-specific linkage tags, that enable the same user to generate unique, unlinkable tags for every context. This security feature holds even for blacklisted users, by having linkage tags in context-specific blacklists generated in a secure multiparty computation manner, where no single server can compute a linkage tag for one context using the linkage tag from another context.
   B. *Unlinkability among user's pseudonymous identities*, i.e. preventing any malicious party from linking multiple pseudonymous accounts to the same holder. This is achieved through the decoupling of linkage tags from the pseudonymous accounts, based on the feature of *Selective-Disclosure*. Moreover, to avoid any statistical and timing attacks that could be used to correlate pseudonyms and linkage tags, the tags are first sent to a mixnet run by the IAMC before reaching the end-service.
2. **Sybil-Resistance:** in other words the capability of the end-service to securely enforce one person - one vote policies, is achieved through the use of linkage tags that are bound to one's personhood token, and are unique per context of usage. This means that we still allow the same person to hold multiple digital identities within the same platform, but the tags generated are person-wise unique, therefore preventing the misuse of these identities for bypassing policies based on the one person - one action principle.
3. **Accountability:** which implies denial of service to misbehaving users, is enforced through the use of blacklists. These blacklists, rely on the same context-linkage tags used for Sybil-resistance described earlier, and therefore are still enforced on a per-person basis. This means that if a person misbehaves on one of their pseudonymous identities, they will be denied future access from all other pseudonymous identities they hold or might create as well. Additionally, using context-specific linkage tags as blacklist entries, implies that blacklists are context-specific as well, making activity tracking impossible even in the case of misbehaving users. Only IAMC nodes in a multi-party fashion can transform a blacklist from one context to the other, further preserving unlinkability.

3.4 System Model and Architecture

Let $\mathcal{U}$ denote the set of users for our system. Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ denote a secret-public key relation. In the standard scenario, we assume that a user $u \in \mathcal{U}$, has successfully participated in a PoP Party(❶), implying that they hold a PoP Token in the form of a public/private key-pair $(pk_u, sk_u) \in \mathcal{R}$ (⓿), with the public share being published in the public party transcript $\mathcal{L}_{\text{Party}}$ by the PoP Party organizers, i.e. $pk_u \in \mathcal{L}_{\text{Party}}$

(❷).

The Credential: We model our system according to the definition of a verifiable credential provided by the W3C specification [19]. A credential is a set of one or more claims made by the same entity. Optionally, credentials might include an identifier and metadata which describe properties of the credential, such as the issuer, validity period etc. A verifiable credential, is a credential together with tamper-evident cryptographic proof(s) on the validity of metadata and claims included in the credential, produced by the issuer. We are now ready to provide a formal definition of the credential which will be referenced throughout this work.

Definition 3 (Credential). A credential is a 3-tuple

$$\texttt{cred} = \{\texttt{metadata}, \mathcal{C}, \sigma\}$$

where:

1. $\texttt{metadata}$ describes the metadata of the credential, i.e. a set of details regarding the use-case and context of usage of the credential, described by any data-type.
2. $\mathcal{C}$ denotes the set of claims embedded in the credential. Moreover, $\mathcal{C} = \mathcal{C}_{\texttt{pub}} \cup \mathcal{C}_{\texttt{priv}}$, where $\mathcal{C}_{\texttt{pub}}$ describes the set of claims with public attributes $\texttt{attr}$, i.e. publicly disclosed values, $\texttt{val}$, corresponding to these attributes, and $\mathcal{C}_{\texttt{priv}}$ describes the set of claims with private attributes, $\texttt{attr}$, containing predicates ϕ, regarding these attributes, together with zero-knowledge proofs, π_ϕ, on these predicates. Optionally, claims might include their $\texttt{provider}$. Hence,
   - if $\texttt{claim}_i \in \mathcal{C}_{\texttt{pub}}$, then

   $$\texttt{claim}_i = \{\texttt{attr}_i, \texttt{val}_i, \texttt{provider}_i\}$$

   - while if $\overline{\texttt{claim}}_i \in \mathcal{C}_{\texttt{priv}}$, then

   $$\overline{\texttt{claim}}_i = \{\texttt{attr}_i, \phi_i, \pi_{\phi_i}, \texttt{provider}_i\}$$

3. σ: the signature issued by the issuer over the metadata and claims embedded in the credential.

For the sake of simplicity we consider a fixed issuer, period of validity and context of usage in our demonstrations, hence we omit the metadata field from our notation. Moreover, we explicitly specify on each occasion the set of public and private claims, leading to the following notation:

$$\texttt{cred} = \{\mathcal{C}_{\texttt{priv}}, \mathcal{C}_{\texttt{pub}}, \sigma\}$$

Credential Issuance: The lifetime of a 3PBCS credential begins with the user requesting a credential based on a set of claims. We follow the threshold-issuance approach of *Coconut* for providing users with credentials. In 3PBCS we delegate the threshold-issuance functionality to the IAMC.

The user starts by selecting the desired attributes to be embedded in the credential (❸). Depending on the use case, a subset of these attributes can be treated as private i.e., never disclosing their values in clear, but instead proving a predicate about these values to the issuing authorities in a Zero-Knowledge manner. Such functionality is inherited from *Coconut.* To better illustrate this feature, a user might request a credential where one of the claims describes the age attribute, and instead of publicly disclosing the numerical value representing the age, they can simply send to the authorities a zero knowledge proof proving the statement "$0 \leq \texttt{age} \leq 100$".

In 3PBCS we leverage this feature to allow the use of legacy PoP Tokens, i.e. $(pk_u, sk_u) \in \mathcal{R}$, for bootstrapping the issuance of credentials, transferring Proof-of-Personhood from the user's secret key sk_u, to the crendetial itself. This is done by treating the user's secret key sk_u, as a mandatory private attribute to be embedded into one of the credential's claims. Upon requesting the credential, the user is expected to prove in a zero-knowledge manner to the issuing authorities, i.e. IAMC nodes, that their key sk_u, indeed corresponds to a public key pk_u out of some trusted PoP Party transcript.

Once the user has prepared her public and private claims she is ready to forward the credential request to the IAMC (❹). Note that private claims are sent together with zero-knowledge proofs proving that they conform to application-specific predicates. In addition to application-specific predicates, a 3PBCS credential request must always contain a proof about the validity of the secret key claim as a PoP Token.

This means that each of the IAMC nodes handling a credential request from some user with secret key sk_u, has to validate the following proof from the user, using the PoP Party transcript $\mathcal{L}_{\text{Party}}$:

$$\pi_\nu = ZKPK\left\{sk_u : \vee_{pk_i \in \mathcal{L}_{\text{Party}}} \left((sk_u, pk_i) \in \mathcal{R}\right)\right\}$$

Once the IAMC node has verified the validity of the user's personhood like described above, along with other application specific criteria for the issuance of the credential (❺), it sends back to the user a partial credential signature $\tilde{\sigma}$(❻). Upon receiving a threshold of such partial signatures, the user is ready to aggregate them, forming a full credential signature σ (❻').

Note that the user is not restricted in requesting only one credential per secret-key. Instead, a user is free to request as many credentials as desired, where each of them could represent a different pseudonymous identity. This means that the same secret key sk_u can be found inside two different credentials representing a separate pseudonymous account each:

$$\mathtt{cred}_A = \{\overline{sk_u}, \mathtt{nym}_A, \sigma_A\} \text{ and } \mathtt{cred}_B = \{\overline{sk_u}, \mathtt{nym}_B, \sigma_B\}$$

A detailed description of the issuance process will be provided in the next section.

Using the Credential: Assuming now that a user has obtained a credential $\mathtt{cred} = \{\mathcal{C}_{\mathtt{priv}}, \mathcal{C}_{\mathtt{pub}}, \sigma\}$ from the IAMC, where $\mathcal{C}_{\mathtt{priv}} \ni \overline{\mathtt{claim}_i} = \{\mathtt{SecretKey}, \text{`}\mathtt{Valid\ PoP\ Token}\text{'}, \pi_\nu, \mathcal{L}_{\mathrm{Party}}\}$, we further describe how the user might use this credential for authorization to an application-specific context $\mathtt{ctx}$, while guaranteeing accountability and Sybil-Resistance for the third party service.

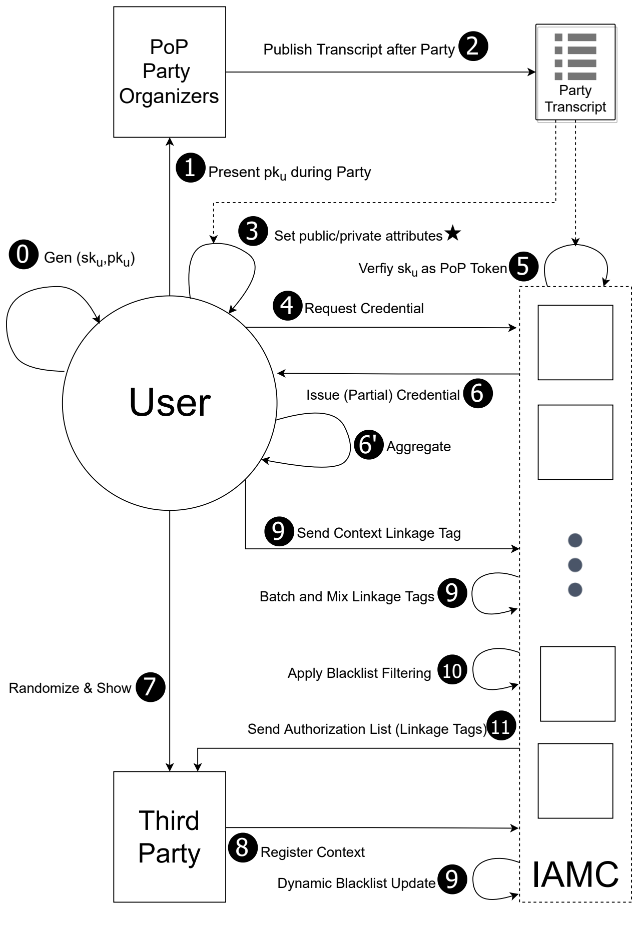
PoP Party Organizers
Publish Transcript after Party 2
Party Transcript
1 Present pk$_u$ during Party
3 Set public/private attributes★
0 Gen (sk$_u$,pk$_u$)
Verfiy sk$_u$ as PoP Token 5
4 Request Credential
User
Issue (Partial) Credential 6
6' Aggregate
9 Send Context Linkage Tag
Batch and Mix Linkage Tags 9
Apply Blacklist Filtering 10
Randomize & Show 7
Send Authorization List (Linkage Tags) 11
Third Party
8 Register Context
Dynamic Blacklist Update 9
IAMC

Fig. 4. High-level overview of 3PBCS.
($\star$) : sk_u is a mandatory private attribute for the requested credential.

The user starts by choosing the attributes she wants to disclose to the service depending on the use case. This is enabled by the feature of selective-disclosure inherited from *Coconut*, and enables the user to regroup their claims between $\mathcal{C}_{\mathtt{pub}}$ and $\mathcal{C}_{\mathtt{priv}}$. Similar to credential issuance, private attributes selected for disclosure to a third party verifier, can be accompanied with proofs on application specific statements for enforcing different authorization policies without revealing data publicly. Moreover, these predicates do not have to be the same as those verified during issuance of the credential. For instance, this time the user might want to prove to a third-party service the statement "$\mathtt{age} \geq 18$" for the age attribute seen earlier. While the user is free to re-group public and private claims depending on the use-case, 3PBCS requires that the secret key attribute is always treated as a private claim, and never disclosed in public.

Upon redistributing her claims into the new sets $\mathcal{C}'_{\mathtt{pub}}$ and $\mathcal{C}'_{\mathtt{priv}}$, the user re-randomizes her signature σ into σ' in *Coconut* fashion, and sends the credential $\mathtt{cred}_{\mathtt{pub}} = \{\mathcal{C}'_{\mathtt{priv}}, \mathcal{C}'_{\mathtt{pub}}, \sigma'\}$ to the verifier (❼) together with the request representing their desired action.

Contexts of Usage: We denote by $\mathcal{A}$ the set of possible actions to be taken by a user within a third-party service using the credential. Then, we categorize this set into two different sets of actions $\mathcal{A}_{\mathtt{priv}}$ and $\mathcal{A}_{\mathtt{pub}}$ where:

– $\mathcal{A}_{\mathtt{priv}}$: denotes the set of actions that are unique per pseudonymous account,
– $\mathcal{A}_{\mathtt{pub}}$: denotes the set of actions that are available to be carried by multiple pseudonymous accounts.
– $\mathcal{A} = \mathcal{A}_{\mathtt{priv}} \cup \mathcal{A}_{\mathtt{pub}}$.

To illustrate the difference between these sets, consider the example of user *alice* creating a new post on a social media platform, on which users *bob* and *charlie* cast a like or vote to. Then, the action of creating the post in the first place would belong to $\mathcal{A}_{\mathtt{priv}}$ since neither *charlie* nor *bob* can re-create the exact same post. However, the action of casting a like or vote to this post would be in $\mathcal{A}_{\mathtt{pub}}$, as both *bob* and *charlie* (as well as any other user from their pseudonymous account) can execute the same exact request for this action.

Linkage Tags: A user on 3PBCS is requested to generate a context-specific linkage tag for each action, represented by an application-specific context, $\mathtt{ctx}$. This tag is computed using the user's personhood-bound secret key sk_u. Moreover, given that sk_u is treated as a private attribute in the credential, the user can provide a zero-knowledge proof where the predicate to be proven is the correct computation of this linkage tag.

Hence, if we let $\theta(\texttt{ctx})$ be a context-specific value computed by the pseudo-random function $\theta(\cdot)$, which will be defined later in our specification, for each action represented by the application-specific context ctx, the user u has to provide to the service a linkage tag

$$L_u^{ctx} = \theta(\texttt{ctx})^{sk_u},$$

together with a Zero-Knowledge proof on the correct computation of this tag based on the secret-key embedded in the user's credential:

$$\pi = ZKPK\left\{sk_u : L = \theta(\texttt{ctx})^{sk_u}\right\}$$

Assuming that $\theta(\cdot)$ is deterministic, and recalling that a user can obtain a single PoP Token (i.e., a unique sk_u), this will force the user to create a unique linkage tag L_u^{ctx} per each context. The third-party service then, can use this uniqueness property of the provided tags, in order to ensure Sybil-Resistance. Hence, a person holder of two pseudonymous accounts $\texttt{cred}_A = \{\overline{sk_u}, \texttt{nym}_A, \sigma_A\}$ and $\texttt{cred}_B = \{\overline{sk_u}, \texttt{nym}_B, \sigma_B\}$ can use both these accounts to cast a vote or like represented by context ctx $\in \mathcal{A}_{\texttt{pub}}$, but they will be forced to generate the same linkage tag $L_u^{\texttt{ctx}}$. At the end, the end-service will only count unique linkage tags for calculating the number of votes, likes, follows etc.

An immediate privacy concern that arises here, is that if the user sends their request from their first account as a tuple $(\texttt{cred}_A = \{\overline{sk_u}, \texttt{nym}_A, \sigma_A\}, \texttt{ctx}, L_u^{\texttt{ctx}})$, and later from their second account as $(\texttt{cred}_B = \{\overline{sk_u}, \texttt{nym}_B, \sigma_B\}, \texttt{ctx}, L_u^{\texttt{ctx}})$ directly to the service, a malicious third-party service can immediately link $\texttt{nym}_A$ and $\texttt{nym}_B$ with the linkage tag $L_u^{\texttt{ctx}}$, learning that the same person holds the two accounts. Note that this issue arises only for contexts describing above-defined public actions $\mathcal{A}_{\texttt{pub}}$, i.e. only if $\texttt{ctx} \in \mathcal{A}_{\texttt{pub}}$.

To avoid this issue, in 3PBCS, instead of sending the linkage tags directly to the end-service, users send linkage tags together with a partial credential to the IAMC for batching and mixing (❾). The IAMC nodes serve as a mix-net, shuffling linkage tags for each context, and then forwarding them to the end-service at the end of each epoch (⓫). Upon receiving the linkage tags, the end-service is able to verify their validity, and use them to update its count on the corresponding public action.

To summarise, the user first sends $(\texttt{cred}_{\texttt{pub}} = \{\overline{sk_u}, \texttt{nym}_A, \sigma_A\}, \texttt{ctx})$ to the end-service as described earlier in step (❼). In this example we treat the user's pseudonym $\texttt{nym}_A$ as a one of the credential attributes, which in this first disclosure is sent in public. Hence, the end-service can immediately add $\texttt{nym}_A$ in the list of voters, even though the vote count will not be updated until the reception of the user's linkage tag for this context. Secondly, the user sends the tuple $(\texttt{cred}_{\texttt{priv}} = \{\overline{sk_u}, \overline{\texttt{nym}_A}, \sigma_A\}, \texttt{ctx}, L_u^{\texttt{ctx}})$ to the IAMC for batching and mixing as described in step (❾). Note that the first message does not contain a linkage tag, while the second does not contain a pseudonym as part of the public attributes. Moreover, each signature σ on the credential is re-randomized every time it is shown based on the *Coconut* scheme.

By first batching and then mixing linkage tags for the same context, the IAMC removes all correlation between pseudonyms and linkage tags that reach the end-service. This preserves Sybil-Resistance, since the end-service still receives verifiable linkage tags with the corresponding proofs of computation included in the partial credential, while allowing the same user, to use multiple pseudonymous identities for taking public actions.

Blacklisting: Apart from serving the purpose of Sybil-protection, the above-described linkage tags in 3PBCS serve as the entries of blacklists. Users on our scheme are blacklisted only if they are reported by a threshold of other users in a way that the IAMC can verify. Moreover, we note that reports can be only filed for private actions carried under one of the user's pseudonyms, i.e. actions described by $\texttt{ctx} \in \mathcal{A}_{\texttt{priv}}$.

Once the threshold of reports has been filed to the IAMC for an action described by $\texttt{ctx}_i$, the latter adds to its distributed blacklist hashtable the following entry: $\texttt{blacklist[ctx}_i\texttt{]+=}L_u^{\texttt{ctx}_i}$. Next, for any subsequent action to be carried by a user, the end-service will register new context for this action $\texttt{ctx}_j$ where $j > i$, with the IAMC (❽). Upon receiving the registration for the new context, IAMC nodes will add a new blacklist entry for this user under $\texttt{ctx}_j$ by collectively computing $L_u^{\texttt{ctx}_j} = F_{\texttt{MPC}}(L_u^{\texttt{ctx}_i}, j)$ and adding it to distributed blacklist hashtable $\texttt{blacklist[ctx}_j\texttt{]+=}\ L_u^{\texttt{ctx}_j}$ (❾). Note that the function $F(\cdot,\cdot)$ must be only possible to compute in secure multiparty computation manner, preventing a single node from the IAMC from learning its value independently. Otherwise, each node in the IAMC would easily exploit this linkage update mechanism for tracking user activity.

Then, if a blacklisted user u from context $\texttt{ctx}_i$ would attempt to take an action for a new context $\texttt{ctx}_j$, $j > i$, the exit IAMC node, responsible for forwarding the linkage tag list to the end-service, would easily omit the linkage tag $L_u^{\mathtt{ctx}_j}$ from this list, by performing a lookup on `blacklist[ctx`$_j$`]` (10).

4 The 3PB Credential Scheme

In this section we describe the 3PB credential scheme in its full technical specification. For the credential issuance and verification steps, we describe the extension of the *Coconut* scheme, wrapping around its procedures, omitting technical details, which are already described in [8].

4.1 General Setting and Notation

Let the Identity and Accountability Management Authority (IAMC) be composed of n nodes represented by the set $\mathcal{S}$, each denoted by $node_i, i \in [1, \ldots, n]$.

Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ denote a secret-public key relation. As mentioned earlier, we assume that a user $u \in \mathcal{U}$, holder of a key pair $(sk_u, pk_u) \in \mathcal{R}$ has participated in a PoP Party, submitting pk_u, and the party transcript for the event, $\mathcal{L}_{\text{Party}}$, containing all public keys of participants, has been published by the Party Organizers, and is accessible via a trusted channel to each $node_i$ in the IAMC.

Let $t \in \mathbb{Z}^+, t \leq n$ denote the threshold parameter indicating the minimum number of IAMC nodes that need to sign a user credential request before the user can retrieve the full credential. Moreover, let q denote the maximum number of credential attributes, including private and public, supported by the system.

Let M with $|M| \leq q$ denote the set of all attribute values that the user wishes to include in their credential. The attributes corresponding to these values, might be either public or private, defining public and private claims $\mathcal{C}_{\text{pub}}$ and $\mathcal{C}_{\mathtt{priv}}$ respectively. We denote by M_{pub} the set of values from public attributes and $M_{\mathtt{priv}}$ the set of values from private attributes. While the issuing authorities or verifiers can clearly access $m \in M_{\text{pub}}$, the values $\tilde{m} \in M_{\mathtt{priv}}$ are never transmitted in the clear by the user, instead commitments on these values are used to prove predicates about them. Moreover, we let $\phi(\tilde{m}), \tilde{m} \in M_{\mathtt{priv}}$ denote a predicate on the commitment for the value $\tilde{m}$, that needs to be proven *to the issuing authorities during issuance* of the credential, while we denote by $\phi'(\tilde{m}), \tilde{m} \in M_{\mathtt{priv}}$ a predicate on the commitment for the value $\tilde{m}$, that needs to be proven *to the verifier during verification* of the credential. Both types of predicates can be application-specific. Moreover, note that while the set of overall attribute values M remains invariant, the user might re-distribute values $m \in M$ between M_{pub} and $M_{\mathtt{priv}}$ every time they send the credential to an issuing authority or verifier.

4.2 The Scheme

4.2.1 Initialization

◊ **Setup**$(1^\lambda, q) \longrightarrow (params)$: Invoke the *Coconut*.**Setup** procedure by setting

$$params \longleftarrow Coconut.\mathbf{Setup}(1^\lambda, q)$$

Then,

$$params = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, h_1, h_2, \ldots, h_q)$$

Where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ denotes a bilinear group (pairing of type 3) of order p, where p is an λ-bit prime, where $\mathbb{G}_1 = \langle g_1 \rangle = \langle h_1 \rangle = \langle h_2 \rangle = \cdots = \langle h_q \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$.

After setting up all necessary parameters we are ready to engage in the key generation step for IAMC nodes. There are three types of keys that these nodes need to be provided with: *Coconut* keys, keys for their mix-server functionality, as well as a secret seed for re-randomizing context secret-keys in a multiparty-fashion.

◊ **IAMC.Init**$(params, t, n, q) \rightarrow (sk, vk, pk, seed)$: First each IAMC node engages in the Coconut.**TTPKeyGen**$(params, t, n, q)$ procedure, which requires either execution from a trusted third party, or a distributed key generation (DKG) step among these nodes. After Coconut.**TTPKeyGen**$(params, t, n, q)$ has been completed, each IAMC node $i \in [1, \ldots, n]$ has obtained a secret key

$$sk_i = (x_i, y_{i_1}, \ldots, y_{i_q}) = (v(i), w_1(i), \ldots, w_q(i))$$

and has published their verification key

$$vk_i = (g_2, \alpha_i, \vec{\beta_i}) = (g_2, g_2^{x_i}, (g_2^{y_{i_1}}, \ldots, g_2^{y_{i_q}}))$$

and public key

$$pk_i = (g_1, X_i, \vec{Y_i}) = (g_1, g_1^{x_i}, (g_1^{y_{i_1}}, \ldots, g_1^{y_{i_q}})).$$

Each node $i \in [1, \dots, n]$ picks a random seed

$$seed_i \xleftarrow{\$} \{0,1\}^\lambda.$$

Moreover, each node initializes the local blacklist hashtable $\texttt{blacklist}_i\texttt{[]} = \emptyset$.

4.2.2 Credential Issuance

Once the initialization procedure is completed, the user now is ready to request a credential from the IAMC nodes acting as *Issuing Authorities* in the *Coconut* scheme. The issuing procedure then is described below:

◇ $\textbf{Issue}(M_{\texttt{priv}} \ni sk_u, M_{\texttt{pub}} \ni \texttt{nym}, \phi) \longrightarrow (cred)$: While the user is free to choose their own sets of public and private attributes to be included in the credential, it is mandatory, that they include the PoP Token secret-key sk_u as part of their private attributes, together with the following predicate $\phi_{PoP}(sk_u)$: "sk_u `corresponds to at least one public key` $pk_i \in \mathcal{L}_{\text{Party}}$". This predicate, can be proven in Zero-Knowledge manner as follows:

$$\pi_{\phi_{PoP}} = NIZK \left\{ sk_u : \bigvee_{pk_i \in \mathcal{L}_{\text{Party}}} ((sk_u, pk_i) \in \mathcal{R}) \right\}$$

Moreover, in 3PBCS, the user is requested to include in the set of public attributes they use for requesting a credential, their desired pseudonym $\texttt{nym}$. Let ϕ_{App} denote the set of application-specific predicates required to be proven for the credential on one or more private attributes $\tilde{m}$. Then, by setting the compound predicate

$$\phi = \phi_{PoP} \wedge \phi_{App}(\tilde{m}_1, \dots, \tilde{m}_q)$$

the user engages in the *Coconut*.**IssueCred** procedure with a threshold of IAMC nodes, which describes the issuance and aggregation of partial signatures:

$$\sigma_i \longleftarrow \textit{Coconut}.\textbf{IssueCred}(M_{\texttt{priv}} \cup M_{\texttt{pub}}, \phi)$$

Lastly the user executes *Coconut*.**AggCred** to obtain the full credential signature using the partial signatures obtained:

$$\sigma \longleftarrow \textit{Coconut}.\textbf{AggCred}(\sigma_1, \dots, \sigma_t)$$

and finally acquires the credential:

$$\texttt{cred} = \{\overline{sk_u}, \texttt{nym}, \sigma\}$$

Claims other than the secret key sk_u and user's pseudonym $\texttt{nym}$ are omitted in the notation above, but remain in the credential. Claims with an overbar indicate private attributes.

4.2.3 Using the Credential

After describing the credential issuance procedure, we now show how a user might use the obtained credential to perform actions on a third-party service, with unique identifier `sID`.

In general, for authentication and authorization through the use of the credential, our scheme relies on the two procedures *Coconut*.**ProveCred** (run by the user) and *Coconut*.**VerifyCred** (run by the verifier) described in *Coconut*. We first explain the inputs and outputs of these two procedures and later see how we adapt their use within 3PBCS:

◇ *Coconut*.$\textbf{ProveCred}(vk_0, M_{\texttt{prv}}, \sigma, \phi') \rightarrow (\sigma', \Theta, \phi')$: Inputs used by the *Coconut*.**ProveCred** procedure, are described as follows:

1. vk_0 : the verification key published during initialization.
2. $M_{\texttt{priv}}$: the set of *private attributes* that the user has chosen for this disclosure. Note that $M_{\texttt{priv}}$ used in this case might contain attributes that were treated as public in the issuance step, but the user does not want to disclose to the verifier during this use. Likewise, certain attributes that were treated as private during issuance, now might not be in $M_{\texttt{priv}}$ any longer, in case the user decides to disclose them publicly to the verifier.
3. ϕ' : a compound (application-specific) predicate to be proven to the verifier regarding the committed values from $M_{\texttt{priv}}$. Like the set of private attributes itself, this set of predicates do not have to be the same as those used during issuance.
4. σ : the credential signature received during issuance as described earlier.

The procedure, then, upon executing on this set of inputs produces the following outputs:

1. σ' : a re-randomized instance of the original credential signature σ. Both are verified using exactly the same procedure, thanks to properties of pairing-based signatures such as Pointcheval-Sanders.
2. Θ : describes a composite object containing the following values:
   – commitments on the values corresponding to the private attributes in $M_{\texttt{priv}}$,
   – Non-Interactive ZKP, proving (a) the correct computations of these commitments, as well as (b) the validity of the compound predicate ϕ' about the committed values on $M_{\texttt{priv}}$
3. ϕ' : the original compound predicate about the committed values from the input.

◇ *Coconut.*$\mathbf{VerifyCred}(vk_0, \Theta, \sigma', \phi') \rightarrow True/False$: This procedure, simply takes as an input the outputs from the *Coconut.***ProveCred** procedure above, verifies the $NIZK$ inside Θ against the commitment on the private attributes and the predicate ϕ' on these attributes.
Lastly, the verifier checks the signature σ' on the credential against the committed values, and the verification key vk_0 from the issuing authorities.
If everything verifies successfully, the procedure returns $True$, and $False$ otherwise.

Assume now, that the user wants to perform some action, `actn` within the application with identifier `sID`. First, the user chooses the relevant attributes for `actn`, both private and public, M_{prv} and M_{pub} respectively. The procedure then is described as follows:

◇ $\mathbf{PerformAction}(\texttt{actn}, M_{\text{prv}}, M_{\text{pub}}, \sigma)$: As a first step the user prepares a credential including all application and context-specific attributes, grouped between private and public as $M^{\texttt{srv}}_{\text{pub}}$ and $M^{\texttt{srv}}_{\text{prv}}$, together with application specific predicates $\phi'_{\texttt{srv}}$, and sends it together with the desired action to the end-service.

◇ $\mathbf{ProveCredential}(\texttt{actn}, M_{\text{prv}}, \sigma, \phi')$: The user prepares

$$\begin{aligned}\texttt{cred}_{\texttt{srv}} &= \left\{\mathbf{ProveCred}(vk_0, M^{\texttt{srv}}_{\text{prv}}, \sigma, \phi'_{\texttt{srv}}), M^{\texttt{srv}}_{\text{pub}}\right\} \\ &= \left\{\{M^{\texttt{srv}}_{\text{prv}}, \Theta_{\texttt{srv}}, \phi'_{\texttt{srv}}\}, M^{\texttt{srv}}_{\text{pub}}, \sigma'_{\texttt{srv}}\right\}\end{aligned}$$

The user sends ($\texttt{cred}_{\texttt{srv}}$,`actn`) to the end-service, which process the request as follows:

◇ $\mathbf{VerifyCredential}(vk_0, \texttt{cred}_{\texttt{srv}}, \texttt{actn})$: First, the third-party service verifies the following:
1. The predicates $\phi'_{\texttt{srv}}$ received inside the credential, comply with those pre-defined for the desired action *actn*.
2. *Coconut.*$\mathbf{VerifyCred}(vk_0, \texttt{cred}_{\texttt{srv}})$ returns $True$.

These conditions ensure to the third-party service that the user is eligible for performing the requested action.

If any of the above conditions fail, the third-party service denies the request returning $\perp$.

Depending on the type of the action `actn`, the third-party service proceeds as follows:
∗ *If* `actn`$\in \mathcal{A}_{\texttt{priv}}$, meaning it is a single-user action, or `actn`$\in \mathcal{A}_{\text{pub}}$, meaning it is a multi-user action and has not been registered with the IAMC in the past, the service generates a new context identifier for the action, $\texttt{ctx}_k$, and invokes $\mathbf{RegisterCotext}(\texttt{ctx}_k)$ from the IAMC.
∗ *Else, if* `actn` has been registered with the IAMC already, the third-party service fetches the respective context identifier $\texttt{ctx}_k$ from its local storage.

The third-party service then, returns the context identifier $\texttt{ctx}_k$ to the user.

◇ $\mathbf{RegisterContext}(\texttt{ctx}_k)$: This procedure is run by each of the IAMC nodes, $\texttt{node}_i, i \in [1, \ldots, n]$ when the third-party service wants to register a new context with the IAMC.
First, the third-party service, sends an authenticated message containing the context identifier $\texttt{ctx}_n$ in a broadcast fashion to the IAMC nodes. We omit details of the broadcast protocol itself, but we put emphasis on the fact that the messages must be verified for authenticity by each node receiving them.
Moreover, the nodes need to be running a consensus algorithm on the order they process each new context identifier. This is easily achievable and we do not provide further details here. Instead we denote the collectively agreed order of a given context identifier by its subscript, i.e. $\texttt{ctx}_k$. Then upon receiving a new context

identifier, $\mathtt{ctx}_k$, each node i proceeds as follows:

Let

$$R_k^i = PRNG(seed_i, \mathtt{ctx}_k)$$

Then,

- $*$ *If* $k = 0$, i.e. it is the first registered context, set

$$sk_k^i = R_k^i$$

- $*$ *Else*, set

$$sk_k^i = R_k^i * sk_{k-1}, \text{ where } sk_k = \sum_{i\in[n]} sk_k^i$$

Then the public key share for node i for context $\mathtt{ctx}_k$ will be:

$$pk_k^i = h_{\mathtt{sID}}^{sk_k^i}$$

where $h_{\mathtt{sID}} = H_{\mathbb{G}_1}(\mathtt{sID})$ and $H_{\mathbb{G}_1}(\cdot)$ is a cryptographic hash function mapping to elements of the group $\mathbb{G}_1$.

Lastly, the shared public key for context $\mathtt{ctx}_k$, is

$$pk_k = h_{\mathtt{sID}}^{sk_k} = \prod_{i\in[n]} pk_k^i$$

Note that $pk_k^i = pk_{k-1}^{R_k^i}$, and therefore no party needs to learn the common shared secret key of any context at any point in time.
Next, the user who has now received the context identifier $\mathtt{ctx}_k$ from the third-party service after the **VerifyCredential** procedure, queries the IAMC nodes for their public key shares pk_k^i for this context. Upon receiving all such shares, the user can compute the shared public key $pk_k = \prod_{i\in[n]} pk_k^i$.

◊ **ProvideLinkageTag**$(\mathtt{ctx}_k, pk_k, sk_u, \sigma)$: First the user computes a context-specific linkage tag, using the public key for this context derived from IAMC nodes, and the user's secret-key sk_u.

$$L_k^u = pk_k^{sk_u}$$

Next, the user, prepares a credential that contains a single private claim

$$\overline{\mathtt{claim}}_{sk_u} : \{\overline{sk_u}, \phi'_L, \pi_{\phi'_L}, \sigma'_L\}$$

where

ϕ'_L: "L_k^u `was properly computed using` sk_u"

and

$$\pi_{\phi_L} = NIZK\left\{sk_u : L_k^u = pk_k^{sk_u}\right\}$$

whereas σ'_L is a re-randomization of the signature σ received upon issuance of the credential. This can be done using the feature of *Selective-Disclosure* in the *Coconut*.**ProveCred** procedure described above, setting all attributes as private, i.e. $M = M_{\text{prv}}$, and using ϕ'_L as described above as a single predicate:

$$\begin{aligned}\mathtt{cred}_{\text{anon}} =& \left\{\textbf{ProveCred}(vk_0, M_{\text{prv}}, \sigma, \phi'_L), M_{\text{pub}}\right\}\\ =& \left\{\{M_{\text{prv}}, \Theta_L, \phi'_L\}, M_{\text{pub}} = \emptyset, \sigma'_L\right\}\end{aligned}$$

Note that the credential above does not contain any information on the user, apart from the fact that they hold a legitimately signed credential, and that the secret-key sk_u embedded in this credential has been used to compute L_k^u.

Using the anonymous credential prepared and the linkage tag computed,the user composes the following message object:

$$Tag_k^u = \{\mathtt{ctx}_k, \mathtt{cred}_{\text{anon}}, L_k^u\}$$

Public and Private Actions: As described earlier, we categorize the entire set of possible actions $\mathcal{A}$, that a user might request to take within a third-party service as public, $\mathcal{A}_{\text{pub}}$ and private,$\mathcal{A}_{\mathtt{priv}}$. Private actions, denote actions that are meant to be executed only by one user at a time, meaning that a new context identifier $\mathtt{ctx}_k$ will be issued and registered by the third-party service each time a user requests to perform such action. This implies that linkage tags L_k^u and context identifiers $\mathtt{ctx}_k$ are in a one-to-one relation for $\mathtt{ctx}_k \in \mathcal{A}_{\mathtt{priv}}$, i.e. only one linkage tag will ever be created for the context, making it impossible for two accounts belonging to the same user, to yield the same linkage tag. A simple example is creating a new post on a social network, where for each new post the social network service will create a unique identifier, that is bound to the account that was used to generate the post.

Public actions on the other side, such as votes, likes, etc., allow executions by multiple users on the same context identifier, leading to a one-to-many relation between context identifiers and linkage tags. This means that if a person desires to engage in the same action from two or more distinct pseudonymous accounts, e.g. u_1 and u_2, they will be forced to generate the same linkage tag for each of these accounts, i.e. if $\texttt{ctx}_k \in \mathcal{A}_{\texttt{pub}}$ and u_1, u_2 belong to the same person, then $L_k^{u_1} = L_k^{u_2}$, enabling the service to infer the common holder of u_1 and u_2.

Mixing: To bypass the above-described issue, if $\texttt{ctx}_k \in \mathcal{A}_{\texttt{priv}}$, users forward the corresponding tags Tag_k^u to the IAMC, which now acts as a mix-network. This is done to prevent the third-party service from correlating the pseudonymous credentials received during the **ProveCredential** procedure, with the corresponding linkage tags, by performing timing analysis and matching the order of reception. Such a correlation would immediately break user pseudonymity.

Concerning the design of the mix network, IAMC nodes form a layered mixnet architecture of three layers, with entry (IN), first layer (L1) and exit nodes (OUT) on each path. Let the pair-wise disjoint sets $\mathcal{S}_{\texttt{in}}, \mathcal{S}_{\texttt{L1}}, \mathcal{S}_{\texttt{out}} \subset \mathcal{S}$ denote the nodes corresponding to each of these layers respectively. Paths are computed in a *source-routing* manner as follows then:

◊ **SetMixRoute**$(Tag_k^u) \longrightarrow (mix_{\texttt{in}}, mix_{\texttt{L1}}, mix_{\texttt{out}})$: Parse Tag_k^u as $\left\{\texttt{ctx}_k, \texttt{cred}_{\texttt{anon}}, L_k^u\right\}$

1. $mix_{\texttt{in}} \leftarrow \mathcal{H}_{\texttt{mix}}(\texttt{ctx}_\texttt{k})$, where $\mathcal{H}_{\texttt{mix}} : \{0,1\}^\lambda \rightarrow \mathcal{S}_{\texttt{in}}$ is a cryptographic hash function public to all users.
2. $mix_{\texttt{L1}} \xleftarrow{\$} \mathcal{S}_{\texttt{L1}}$.
3. $mix_{\texttt{out}} \xleftarrow{\$} \mathcal{S}_{\texttt{out}}$.

Return $(mix_{\texttt{in}}, mix_{\texttt{L1}}, mix_{\texttt{out}})$.

Moreover, the user performs layered encryption on the message containing the tag, preventing any IAMC node in the path, except for the exit nodes, from learning their value. For this, public keys of IAMC nodes published during initialization are used.

Having tags of the same context sent to unique entry nodes, enables threshold batching: the entry nodes will ensure that they have received a threshold $\tau \geq 2$ of tags for each context, before relaying them to the next node in layer L1. This detail is crucial in achieving the desired privacy guarantees, as supplying the mixnet with a single tag per period and per context would result in an anonymity set of 1, hence no anonymity.

Then for each period T, each entry node collects at least τ encrypted tags per context ctx_k, removes one layer of encryption (revealing the next node), shuffles the tags and relays them to nodes of layer L1. Nodes of layer L1 remove the second layer of encryption, reshuffle the tags, and send them to nodes of the exit layer, which accumulate tags until the end of period T. At the end of the period T, exit nodes remove the last layer of encryption, shuffle the tags, and forward them to the third-party service, which can now take a unique count of tags per each context, for counting verified votes, likes, etc.. The order of the tags received for each context however, appears random to the third-party service, preventing it from connecting linkage tags and pseudonyms received for the action. Therefore, the scheme enables both Sybil-Resistance and *unlinkability* for both user's actions and pseudonymous identities.

Lastly, once the end-service has received the shuffled list of linkage tags for each voter u of context k, i.e. Tag_k^u it is ready to verify the validity of these tags. To be able to do this, the third-party service needs to first collect the partial public key shares from all IAMC nodes for context k, in order to be able to reconstruct the full context public key, i.e. pk_k. Then, using the context public key, the following procedure is invoked to verify the validity of linkage tags:

◊ **VerifyLinkageTag**$(Tag_k^u, pk_k) \rightarrow (True/False)$: First, the third-party service verifies the following, pares Tag_k^u as $\left\{\texttt{ctx}_k, \texttt{cred}_{\texttt{anon}}, L_k^u\right\}$.
Note that the service can successfully verify the predicate
ϕ'_L: "L_k^u `was properly computed using` sk_u"

represented by the zero-knowledge proof-of-knowledge

$$\pi_{\phi_L} = NIZK\{sk_u : L_k^u = pk_k^{sk_u}\}$$

inside $\texttt{cred}_{\texttt{anon}}$, as it has access to both the linkage tag L_k^u and context public key pk_k.
Finally, the service proceeds by verifying the anonymous credential, calling *Coconut*.**VerifyCred**$(vk_0, \texttt{cred}_{\texttt{anon}})$.
If the verification is successful, the service returns *True*, and can use the linkage tag L_k^u in its own Sybil-resistance mechanism.

Otherwise, the service returns *False* and discards L_k^u.

4.2.4 Dynamic Context-Specific Blacklists

Assume now, that a user u has been reported for misbehaviour under some context $\texttt{ctx}_\texttt{i}$. The 3PBC scheme enforces accountability for such malicious actors, by creating blacklists based on reported cases of such misbehaviour. The entries of these blacklists are the same context-specific linkage tags as seen so far. This further implies that blacklists must be also context-specific, therefore we need a secure way to dynamically transform the blacklist from one context to the other. Such an action however, cannot be trusted to a single party or server, since that would immediately enable that party to link linkage tags of the same person under different contexts. For this reason, we only allow the transformation of context-specific blacklists from one context to the other to be performed in a multi-party computation manner by the IAMC, with no single node able to carry the transformation independently on its own.

Reporting: First, to make blacklisting possible, a reporting mechanism must be in place. While the policy of reporting is application and use-case-specific, we assume here that the IAMC is able to receive some proof of misbehaviour from the third-party service, described by an action under a specific context $\texttt{ctx}_\texttt{i}$, and the corresponding linkage tag L_i^u provided by the user. Note here that reporting is only possible for private actions, i.e. $\texttt{ctx}_\texttt{j} \in \mathcal{A}_{\texttt{priv}}$, since public actions involve multiple linkage tags, and are typically either idempotent operations (such as passive information retrieval), or count-based updates (such as likes, votes, etc.). In fact, the main risk in public actions arises from Sybil-attacks, which have been already addressed earlier. Hence, we focus here on misbehaviour that users exhibit on their private actions, such as posting fake news or hate speech for instance.

Once the third-party service is able to prove such misbehaviour, it broadcasts the linkage tag L_j^u provided by the user u together with the reported context identifier $\texttt{ctx}_\texttt{j} \in \mathcal{A}_{\texttt{priv}}$, to invoke the procedure **BlacklistReport**$(L_i^u, \texttt{ctx}_\texttt{i})$ in the IAMC nodes, which we will shortly present. Let $\texttt{ctx}_\texttt{k}, k > j$ be the latest context identifier registered with the IAMC by the third-party service, running the procedure **RegisterContext**$(\texttt{ctx}_\texttt{k})$ described above. Then, upon receiving the report $\{\texttt{ctx}_\texttt{j}, L_j^u\}$, each $node_i \in \mathcal{S}$ proceeds as follows:

```
◊ BlacklistReport($L_j^u$, ctx_j) :
  for $n = 0$; $j + n \leq k$; $n$++:
      blacklist$_i$[ctx$_{j+n}$] $+= L_{j+n}^u$;
      $R_n^i = PRNG(seed_i, \texttt{ctx}_{j+n+1})$;
      broadcast $(L_{j+n}^u)^{R_n^i}$;
      while not $((L_{j+n}^u)^{R_n^s}$ received $\forall s \in \mathcal{S}$):
          wait();
      $L_{j+n+1}^u = \prod_{s\in\mathcal{S}} (L_{j+n}^u)^{R_n^s}$;
```

Additionally, to maintain the blacklist across new contexts being created, the nodes run the following procedure every time a new context $\texttt{ctx}_\texttt{k}$ is registered (by a third-party service) and **RegisterContext**$(\texttt{ctx}_\texttt{k})$ is executed:

```
◊ UpdateBlacklist(ctx_k) :
  $R_k^i = PRNG(seed_i, \texttt{ctx}_k)$;

  for $L$ in blacklist[ctx$_{k-1}$]:
      broadcast $(L)^{R_k^i}$;
      while not $((L)^{R_k^s}$ received $\forall s \in \mathcal{S}$):
          wait();
      blacklist[ctx$_k$] $+= \prod_{s\in\mathcal{S}} (L)^{R_n^s}$;
```

Correctness of Blacklist Entries: We recall that a linkage tag for context $\texttt{ctx}_k$ from user u is computed according to the procedures **RegisterContext** and **ProvideLinkageTag**, described above, where

$$L_k^u = pk_k^{sk_u} = h_{\texttt{sID}}^{sk_u * sk_k} = h_{\texttt{sID}}^{sk_u \sum_{s\in\mathcal{S}} sk_k^s}$$

Moreover, recall that $\forall s \in \mathcal{S}$ we have that $sk_{k+1}^s = R_{k+1}^s * sk_k$, where $R_{k+1}^s = PRNG(seed_i, \texttt{ctx}_{k+1})$, yielding

$$sk_{k+1} = \sum_{s\in\mathcal{S}} (sk_{k+1}^s) = \sum_{s\in\mathcal{S}} (R_{k+1}^s * sk_k) = sk_k \sum_{s\in\mathcal{S}} R_{k+1}^s$$

Note that in procedures **BlacklistReport** and **UpdateBlacklist** above we have

$$\begin{aligned} L_{k+1}^u &= \prod_{s\in\mathcal{S}} (L_k^u)^{R_{k+1}^s} = \prod_{s\in\mathcal{S}} h_{\texttt{sID}}^{sk_u * sk_k * R_{k+1}^s} \\ &= h_{\texttt{sID}}^{sk_u * sk_k * \sum_{s\in\mathcal{S}} R_{k+1}^s} = h_{\texttt{sID}}^{sk_u * sk_{k+1}} \\ &= pk_{k+1}^{sk_u} \end{aligned}$$

Hence, the linkage tag collectively computed by the IAMC nodes, corresponds to the tag that would be computed by the user themselves for context $\mathtt{ctx}_{k+1}$.

Security of Blacklist Entries: Besides correctness of entries, when it comes to blacklist computations, it is crucial to ensure that it is hard for any single entity or server to compute a linkage tag L^u_{k+1} given another linkage tag from a previous context by the same user L^u_k. A single entity being able to compute L^u_{k+1} given L^u_k, can be decomposed in the following attack vectors:

1. A single entity knows the randomization factor $R_{k+1} = \sum_{s \in \mathcal{S}} R^s_{k+1}$ for the secret key sk_{k+1} of context $\mathtt{ctx}_{k+1}$ where $sk_{k+1} = R_{k+1} * sk_k$. To understand why this is hard, we first recall that no single IAMC node $s \in \mathcal{S}$ discloses its secret randomization factors R^s_{k+1} at any time. Instead, all nodes according to the procedure **RegisterContext**, broadcast only the partial public key computed using these randomization factors, i.e. $pk_k^{R^s_{k+1}}$. Then, any party who has collected all the public shares, can take their product to infer $pk_{k+1} = \prod_{s \in \mathcal{S}} pk_k^{R^s_{k+1}} = pk_k^{\sum_{s \in \mathcal{S}} R^s_{k+1}}$. This reduces the capabilities of any potential adversarial party that aims at retrieving the secret randomization factors of IAMC nodes, to solving the CDH problem, which we assume to be hard.
2. A single party can easily access all partial updates for an arbitrary linkage tag, i.e. $(L^u_k)^{R^s_{k+1}} \forall s \in \mathcal{S}$, for any arbitrary context $\mathtt{ctk}_k$ and user u. However, according to our definition of procedures **BlacklistReport** and **UpdateBlacklist**, IAMC nodes will share partial updates for linkage tags, if and only if these linkage tags belong to their local blacklists. Since all partial linkage tag updates are required for a successful computation of consecutive linkage tags in the blacklist, all nodes must collectively agree on the fact that a given linkage tag has been blacklisted before distributing their partial linkage tag updates. Therefore, it is impossible for a single node, to arbitrarily link a given users linkage tag across contexts.

Filtering: Given that all nodes maintain a distributed blacklist hashtable which is dynamically updated as new contexts are registered, and new reports are being filed, exit nodes in our mixnet composition within the IAMC, can use these blacklists to perform preliminary filtering before relaying the accumulated and shuffled linkage tags to the third-party service. This can be easily done by simply excluding all entries in $\mathtt{blacklist}_i[\mathrm{ctx}_k]$ from the list of linkage tags that have been accumulated during period T for context $\mathtt{ctx}_k$. This procedure, will effectively disable the user from taking any actions in the service after they are blacklisted.

4.2.5 Remarks about Relaxed Security Requirements

The full 3PBCS scheme as described above, addresses the three security goals presented in our system specification, namely *Privacy*, *Sybil-Resistance* and *Accountability*. However, in different use cases where *accountability* does not represent a requirement, the scheme can be simplified as follows:

- IAMC nodes need no longer maintain blacklists.
- Verifiers need no longer register contexts for new actions, i.e. the sub-routine **RegisterContext()** does not need to be invoked within the procedure **PerformAction**.
- Users do not have to query the IAMC nodes for shares on the context public keys pk_k, and they can replace the computation of linkage tags in **ProvideLinkageTag()** from

$$L = pk_k^{sk_u}$$

to

$$L = H_{\mathbb{G}_1}(\mathtt{sID} || \mathrm{ctx}_k)^{sk_u}$$

The above reduction of the scheme still enforces Sybil-resistance in a privacy-preserving manner thanks to the context-specific linkage tags and the mixnet functionality offered by the IAMC, but avoids the computational and communicational overhead of maintaining dynamic blacklists.

5 Prototype Implementation

In this chapter we describe the prototype implementation of the 3PBCS scheme, focusing on a general decomposition of the created components as well as on the major implementation challenges at the current state of advancement.

5.1 Overview

The implementation of a proof-of-concept model for our scheme, served two primary objectives:

1. Demonstrate feasibility in the implementation of the designed system.
2. Provide a preliminary evaluation of the performance in our scheme.

Based on the time allocated for development, as well as on the complexity of the 3PB Credential Scheme, the reduced version of the scheme, described in Section 4.2.5 was used as a reference for this proof-of-concept. The main focus was placed on the implementation and testing of the cryptographic primitives enabling the functioning of our scheme. More generic primitives utilised, such as mix-networks, can be easily adapted within our IAMC component from any existing implementation.

5.2 Current State and Contributions

We implemented a prototype of the reduced 3PB Credential Scheme, as described in 4.2.5 was implemented in `Go`. The code can be accessed under: https://github.com/dedis/ppt-code.

5.2.1 Building Blocks

The main external dependency in our implementation consists of `kyber`, the advanced cryptography library in `Go`, developed by the DEDIS Lab at EPFL.

All pairing-based operations rely on the `bn256` Barreto-Naehrig curve [20] supported in `kyber`. All zero-knowledge proofs-of-knowledge are computed using the native `proof` package in `kyber`.

Below we present the full list of natively implemented packages:

A. *Coconut*
In lack of a suitable existing library for the *Coconut* scheme in `Go`, that would fit with the rest of the cryptographic primitives offered by `kyber`, our implementation provides a fully working package for *Coconut*. This package, is also built using `kyber` to facilitate compatibility and extensibility, for future use by projects similar to 3PBCS.

B. *Proof-of-Personhood*
Package that provides a very lightweight abstraction of Proof-of-Personhood Parties and Proof-of-Personhood Tokens. Based on the definition of our scheme these artefacts would be ideally obtained through a proper Pseudonymous Party setup for strong guarantees. Nevertheless, the package treats PoP Tokens as simply a secret-public key-pair, and the PoP Transcript as a set of public keys where the PoP Token belongs to. This removes any restrictions to the concrete mechanism that is used for Sybil-Protection, supporting alternate variants to Pseudonymous Party setups, as long as these conform to the PoP Token and PoP Transcript structure described above.

C. *ThreePBCS*
The main contribution within the scope of this project, was to implement the cryptographic procedures from the 3PBCS scheme described in Section 4.2. The package is structured based on the different actors participating in the scheme, i.e. `User`, `Issuer` and `Verifier`. This design choice, enables the modular embedding of our scheme within larger systems, which is crucial, since our design consists of a credential system that is to be employed in larger set-ups rather than a standalone component.

All public structures that are produced/processed by the different components in the package implement the `binaryMarshalling/binaryUnmarshalling` interfaces for easy transfers over the wire.

D. *Identity and Accountability Management Cothority*
While the `ThreePBCS` package implements the cryptographic functionality of our design, the `IAMC` package provides the functionality of the Identity and Accountability Management Cothority described in our system. This package, therefore, builds on top of the `ThreePBCS` package, primarily utilizing the issuing functionality, and implements the cothority nodes.

Some of the main helper packages are embedded into the `IAMC` package that serve demonstration purposes are `DKG`, for *Distributed Key Generation* and `P2P`, as a simple network communication protocol.

Full-scheme tests and benchmarks are written for both main packages implemented i.e. `Coconut` and `ThreePBCS`.

5.2.2 Implementation Challenges and Solutions

The first major challenge in the implementation of this proof of concept, was the heavy dependence of our scheme on the *Coconut* scheme. While it was initially decided to build on top of an existing implementation of `Coconut` in `Go` by `nymtech` [21], it was later noticed that this package was not suitable for use in this work due to the following reasons:

1. `nymtech`'s implementation did not build on top of the `kyber` package, but instead used a combination of alternate cryptographic libraries and its own implementations for cryptographic primitives.
2. `nymtech`'s implementation restricted parts of the *coconut* scheme, namely application specific predicates for the issuance and verification of the credential, making it impossible to adapt the scheme to our needs.

These made extensibility of the package hard and too complex for our use case, leading to the native implementation of a new *Coconut* package from scratch, relying on the `kyber` library. This package, addresses the two shortcomings mentioned above, and offers a major contribution within the project. However, due to the restricted time-frame and scope for development, the implementation of *Coconut*, lead to our proof of concept being limited to the reduced scheme as noted earlier.

We faced another major challenge when using the `kyber/proof` package for ZKP. As our scheme relies on bilinear pairing operations and signatures, part of the implementation required the composition of NIZKPs that included commitments from both groups $\mathbb{G}_1$ and $\mathbb{G}_2$ in a pairing. Such a feature was unfortunately not supported in the `kyber.Proof` package, which is designed to initialize each `prover` or `verifier` using a single predefined suite/group.

To overcome this limitation, our implementation includes an adapter, that treats bilinear pairings as a suite on its own, and attempts to postpone curve point initialization up to the moment of the operation when that is possible, in order to be able to detect the group that is being operated on within the pairing. The adapter can be found as `pairingAdapter.go` within our native `Coconut` package.

5.2.3 Final Remarks

At its current state, the core `ThreePBCS` scheme is ready for verification and audit. Additionally a demonstration, on actual running nodes can be shown to be working for the scheme up to issuance, and is currently being extended to the demonstration of the full steps of the scheme including verification.

While the implementation is far from a production stage at present, it can be used for evaluations and experiments fulfilling our two main implementation goals.

6 Evaluation

In this section we present the evaluation of the 3PB Credential Scheme. We focus on the efficiency and performance of the cryptographic primitives implemented across the procedures of our scheme as described in section 4.2.

A. *Cryptographic Primitives*

The most substantial components developed during the implementation of the proof-of-concept for 3PBCS consist on the combination and utilisation of cryptographic primitives that enable the functioning and guarantees of our scheme. For this reason we base our evaluation of efficiency and performance on the procedures corresponding to these primitives.

In the following experiments, communication overhead is not included.

The benchmarks were executed on a Unix-based environment running on an Intel Core i5 CPU (base frequency 1.4 GHZ, Quad-Core) and 8 GB of RAM. In all measurements, we rely on an IAMC consisting of 6 nodes, while the number of total attributes per credential as well as sizes of PoP Transcripts vary based on different test-cases described below. All credential attributes are treated as private to reflect worst-case scenarios. All wall-clock time of execution measurements are generated over a sample of 5000 runs.

a) *Execution Time Benchmarks:* Table 1 provides measurements for each 3PBCS procedure implemented across different maximum number of attributed supported, starting from 5 up to 100. We consider 100 attributes to be a sufficiently large amount of attributes per credential for most realistic use cases. For comparison, a classic Identity Platform such as

Attributes	**5**		**25**		**75**		**100**	
Procedure	μ**(ms)**	σ**(ms)**	μ**(ms)**	σ**(ms)**	μ**(ms)**	σ**(ms)**	μ**(ms)**	σ**(ms)**
PrepareCredRequest	21.66	0.64	38.85	1.03	81.71	3.35	103.99	2.03
ProcessCredRequest	88.7	2.47	172.14	4.45	376.68	12.27	487.82	5.97
ProcessBlindedSignature	0.79	0,04	0.80	0.04	0.80	0.11	0.82	0.04
ProveCredential	2.55	0.18	2.58	0.16	2.61	0.19	2.56	0.14
ProvideLinkageTag	3.78	0.23	3.78	0.21	11.52	0.69	13.89	0.77
VerifyCredential	4.84	0.23	6.69	0.33	3.81	0.27	3.747	0.20
VerifyLinkageTag	5.72	0.24	7.57	0.31	12.36	0.56	14.78	1.00

Table 1. Comparison of the performance of 3PBCS cryptographic components across different numbers of private attributes per credential. The used PoP Transcript consists on 30 participants. The procedures `ProcessCredRequest` and `ProcessBlindedSignarure` that are executed once per each issuer in the IAMC, reflect the cumulative runtime over all node executions.

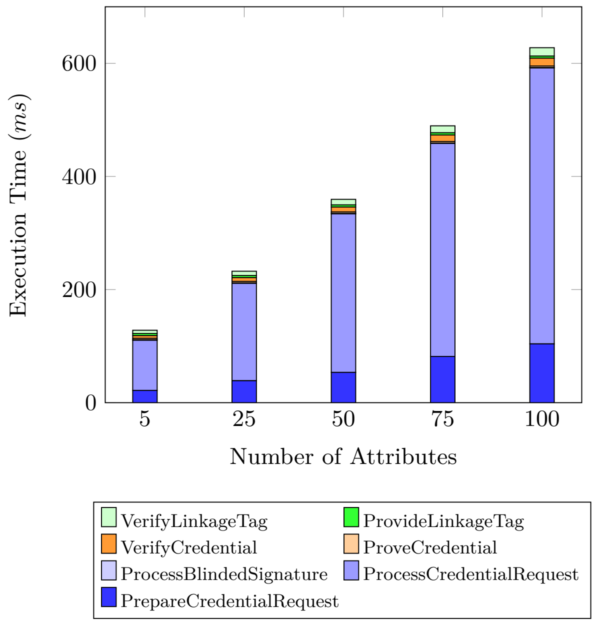

Fig. 5. Chart: Decomposition of total execution time across individual 3PBCS procedures, for varying number of attributes per credential, using an IAMC of 6 servers and a PoP Transcript of 30 participants. The procedures `ProcessCredRequest` and `ProcessBlindedSignarure` that are executed once per each issuer in the IAMC, reflect the cumulative runtime over all node executions. Procedures reflected in blue colors represent credential issuance operations. Procedures reflected in orange colors represent credential verification operations. Procedures reflected in green colors represent operations on linkage tags.

`Auth0` [22] provides about 20 core attributes for each credential.

The experiments reflect a feasible runtime for all main 3PBCS procedures. Procedures wrapped over Coconut routines demonstrate only a slight overhead when compared to the base runtime measurements of the Coconut scheme itself. This is expected due to the additional proofs-of-knowledge introduced in 3PBCS for proving personhood and proper computation of linkage tags. As expected, execution times are in linear dependence to the attributes included in the credential for both issuance and verification. Moreover, the results presented disregard concurrency of computations, therefore procedures run by the IAMC such as `ProcessCredentialRequest` reflect the cumulative runtime over all nodes. In practical terms, the perceived runtime would be considerably lower, subject to network topology and churn. The findings are reflected in Figure 5 for better comprehension.

Moreover, our measurements include varying number of participants in simulated Pseudonymous Party set-ups, reaching up to 1000 members. This would be classified as a medium-large event based on the definition of Ford [11] for Pseudonymous Parties, and therefore serves as a good basis for our measurements, reflecting realistic scenarios. Measuring the effect of this parameter, i.e. total PoP Party participants, is of interest, since our credential issuance directly relies on the size of the PoP Party Transcript in terms of performance. The results are demonstrated in Figure 6.

As noted in Figure 6, overall credential issuance runtime is in a linear relation to the size of the PoP Transcript used. This is expected due to the linear computation and size complexity of zero-knowledge proofs proving the validity of the user's PoP token based on this transcript, similar to the case of Linkable Ring Signatures. This limitation is addressed on Section 8.

b) *Perceived Latency Benchmarks:* Figure 7 presents the perceived latency during credential issuance

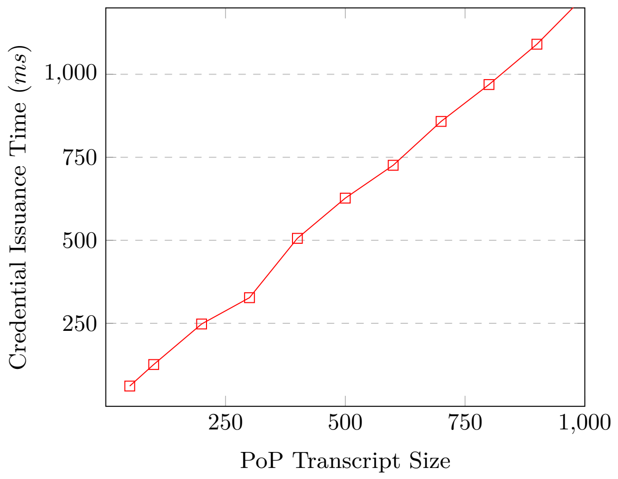

Fig. 6. Effect of PoP Transcript size on credential issuance in 3PBCS.

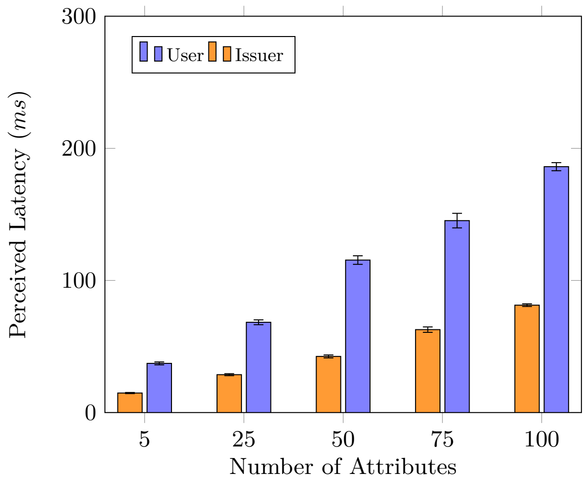

Fig. 7. Chart: Perceived latency for 3PBCS credential issuance between *issuer* and *verifier* actors, for varying number of attributes per credential.We consider a PoP Transcript of 30 participants. IAMC consists on the standard 6-server setup described above.

from both the *User* and *Issuer* actors for the same varying numbers of attributes.

The results show a sub-linear perceived latency for both users and issuers on the number of private attributes that are included in the credential request. While in our previous evaluations we observed a linear such dependence for the overall execution times of isolated procedures, here we note that perceived times are favoured by the concurrent execution of processing credential requests in the IAMC, and therefore these results are expected.

Perceived latency on credential verification is subject to the mix-network topology, churn, and user-generated traffic, therefore we do not include it in our benchmarks.

B. *Complexity*

As part of our evaluation we include the communication and space complexity for running 3PBCS.

Let n denote the number of nodes in the IAMC, and m the number of *private* attributes in the credential. Let q denote the size of the PoP Transcript used for issuance. Let the length of the mix-path be denoted by r. Signature sizes are 128B, and linkage tag sizes 64B. The communication and size complexity decomposition of 3PBCS procedures is reflected in Table 2.

Procedure	Communication	Size
RequestCredential	$\mathcal{O}(n)$	$\mathcal{O}(m+q)$
IssueCredential	$\mathcal{O}(n)$	$\mathcal{O}(m)$
ProveCredential	$\mathcal{O}(1)$	$\mathcal{O}(m)$
ProvideLinkageTag	$\mathcal{O}(n+r)$	$\mathcal{O}(m)$
VerifyCredential	$\mathcal{O}(1)$	$\mathcal{O}(1)$
VerifyLinkageTag	$\mathcal{O}(1)$	$\mathcal{O}(1)$
RegisterContext	$\mathcal{O}(n)$	$\mathcal{O}(1)$
UpdateBlacklist	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$

Table 2. Communication and Size Complexity for 3PBCS procedures. n - number of IAMC nodes; m - number of private credential attributes; q - PoP Transcript Size; r - length of mix-route.

7 Related Work

Anonymous Credential Systems: A wide variety of research [8, 23, 24] has been conducted in the design of anonymous credential systems. These systems enable users to obtain full privacy-control over their attributes, while still being able to obtain and prove the validity of their credential. To the best of our knowledge no such systems have demonstrated how to enforce Sybil-Resistance and accountability in the use of the credentials they provide. In fact, 3PBCS relies on one such scheme as described above, i.e. *Coconut*, and builds on top of it to provide these features.

CanDID: A practical Decentralized Identity (DID) system which enforces both Sybil-Resistance and Accountability, is *CanDID* [25]. This system is one of the first to offer Sybil-resistant and accountable credentials similar to 3PBCS, following a slightly different path in achieving these guarantees. Additionally, CanDID offers usability features such as support for legacy compatibility and key-recovery which 3PBCS lacks at the current state of advancement.

However, one of the potential limitations of CanDID credentials depending on specific use-cases, is that it allows at most one credential per context of usage. By comparison, 3PBCS allows the user to obtain virtually infinite credentials that can be used across different contexts, while still preserving the same guarantees.

Another key difference, is that CanDID bootstraps credential issuance based on DECO [26] - a privacy-preserving oracle protocol that uses cryptographic techniques for enabling users to prove facts about their web (TLS) sessions to oracles while hiding privacy-sensitive data. 3PBCS on the other side, relies on the notion of Proof-of-Personhood, for binding digital identities and physical entities while leaving the choice on how to prove legitimacy for the rest of the claims open to different solutions (including DECO). The first approach offers greater usability, while the second enforces stronger Sybil-Resistance guarantees.

8 Limitations

Here we present the main limitations of the 3PBCS scheme.

8.1 Credential Management

Credential management presents few challenges in the scheme we described so far. These challenges mainly flow from the strong coupling of the credential with the user's proof-of-personhood, which needs to be renewed periodically.

A. *Credential Validity*
The first limitation concerns the lifetime and validity of the credential. Being bound to a PoP Token, i.e. a secret key representing the personhood of the owner, limits the validity of the credential to the validity of this key. In the concrete instantiation we consider throughout this paper for personhood, that is Pseudonym Parties, this would mean that the credential can only be valid until the next in-person gathering. Moreover, since the user will obtain a fresh PoP Token from every new pseudonymous party, this means that their credential will have to be reissued periodically. Unfortunately, as long as Pseudonymous events are used for Sybil-resistance in our scheme, this usability burden has to be accepted as a trade-off for the strong guarantees this mechanism offers. However, 3PBCS is open to being used with alternative methods of proving personhood, which could offer longer periods of validity, and remove the practical barrier of attending in-person events.

Another challenge concerning the validity of the credential, is that a user might attempt to carry Sybil-attacks over time. Since a user will obtain a fresh secret key on each party they attend, and this key represents a random value, it means that all linkage tags computed using this key will be entirely unlinkable with previous keys the user might have used. A first potential solution here would be to require user's devices to store past secret keys, and moreover have end-services only accept votes, likes or follows if the time of their creation lies within the validity period of the user's PoP Token. A second approach, would be to have the user store their own activity, and every time they refresh their proof-of-personhood, to use the fresh secret key for recomputing past linkage tags. In this case, the end-service will discard all metrics that were generated using expired credentials.

B. *Credential Recovery*
At the current stage of advancement our scheme has not treated the issue of credential lost and recovery. We leave this as an issue to be addressed in future work.

8.2 Technical Limitations

The main technical limitation in 3PBCS concerns the blacklisting mechanism presented above. In our scheme blacklist entries are dynamically computed for each context based on entries from previous contexts. This means that 3PBCS is able to guarantee that a user misbehaving on context i can be successfully denied access from all subsequent contexts $j > i$. However, our approach is limited when it comes to the user engaging with actions on existing contexts at the time of misbehaviour, i.e. $k < i$. This means that a blakclisted user for instance will not be able to vote on any new posts after they misbehaviour, but they might still be able to vote on posts from the past.

Another technical restriction arises during credential issuance, where the credential request presented by the user, is linear in the size on the size of the PoP Party transcript. Even though this step is carried only once during the credential's lifetime, it can be mitigated by setting an upper bound on the size of participants to be used in this step. This would effectively mean that the anonymity set of the user is reduced, as a subset of the party transcript is used in this case. However, an optimal choice for this upper bound can be made to minimize the trade-off between communication complexity and privacy. Moreover, thanks to the re-randomizable nature of credentials the effect of reducing the anonymity set during issuance would be minimal to the overall privacy guarantees of the credential.

9 Future Directions

The highest-priority future step for 3PBCS consists of a complete and detailed security analysis for the scheme.

From a design viewpoint, an interesting line of work for 3PBCS would be the optimization of our dynamic blacklisting mechanism, which at present is the most complex component of 3PBCS in terms of communicational overhead. Moreover, functional features such as backward blacklisting and blacklist entry removal presented in our limitations above are worth studying.

In terms of implementation, the development of a full demonstration for 3PBCS will continue after the submission of this work, in order to be able to fully test and benchmark the scheme including the incomplete components presented in section 5.

10 Conclusion

Existing anonymous credential systems to our knoweldge, do not provide support for Sybil-resistance and accountability in their design, two vital guarantees in the way digital identities are used today. This restricts their practical usage on real-world scenarios, where digital metrics have taken a key role, and where accountability is a must. In this thesis work, we have presented the 3PB Credential System, supporting all privacy features of anonymous credential schemes by building on top of *Coconut*, and moreover introducing Sybil-resistance and accountability, while keeping these privacy features intact. We have provided an overview of 3PB Credential System, as well as a detailed description of the system design. Moreover, we have shown the successful implementation of a first prototype for our core scheme. Despite not covering the entire spectre of components described in our design, this implementation has demonstrated feasibility and good performance metrics for the scheme overall.

3PBCS presents a strong candidate for credential systems utilised in different scenarios, starting from social platforms, to digital voting, and anonymous whistleblowing. This is not only due to the above-described security guarantees that it offers, but also due to key usability features. Here, we highlight the capability of users to create virtually infinite digital identities, which is not enabled by any other credential system in a privacy-preserving and fully Sybil-resistant manner, to the best of our knowledge.

11 Acknowledgements

This work was supported and funded by armasuisse Science and Technology. Special thanks to Dr. Daniel Moser and the Cyber-Defence Campus, for their continuous support throughout this project.

References

[1] M. Knell, "The digital revolution and digitalized network society," *Review of Evolutionary Political Economy*, vol. 2, no. 1, pp. 9–25, 2021.

[2] L. Fan, X. Liu, B. Wang, and L. Wang, "Interactivity, engagement, and technology dependence: understanding users' technology utilisation behaviour," *Behaviour & Information Technology*, vol. 36, no. 2, pp. 113–124, 2017.

[3] A. Esfandyari, M. Zignani, S. Gaito, and G. P. Rossi, "User identification across online social networks in practice: Pitfalls and solutions," *Journal of Information Science*, vol. 44, no. 3, pp. 377–391, 2018.

[4] J. Hinds, E. J. Williams, and A. N. Joinson, ""it wouldn't happen to me": Privacy concerns and perspectives following the cambridge analytica scandal," *International Journal of Human-Computer Studies*, vol. 143, p. 102498, 2020.

[5] A. L. Burrow and N. Rainone, "How many likes did i get?: Purpose moderates links between positive social media feedback and self-esteem.," *Journal of Experimental Social Psychology*, vol. 69, pp. 232–236, 2017.

[6] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*, pp. 251–260, Springer, 2002.

[7] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," *Journal of economic perspectives*, vol. 31, no. 2, pp. 211–36, 2017.

[8] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," in *NDSS*, The Internet Society, 2019.

[9] M. Borge, E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Proof-of-personhood: Redemoc-

ratizing permissionless cryptocurrencies," in *EuroS&P Workshops*, pp. 23–26, IEEE, 2017.

[10] J. K. Liu and D. S. Wong, "Linkable ring signatures: Security models and new schemes," in *International Conference on Computational Science and Its Applications*, pp. 614–623, Springer, 2005.

[11] B. Ford, "Identity and personhood in digital democracy: Evaluating inclusion, equality, security, and privacy in pseudonym parties and other proofs of personhood," *CoRR*, vol. abs/2011.02412, 2020.

[12] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," *Technical Report/ETH Zurich, Department of Computer Science*, vol. 260, 1997.

[13] D. Evans, V. Kolesnikov, M. Rosulek, *et al.*, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[14] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *International conference on the theory and application of cryptology and information security*, pp. 552–565, Springer, 2001.

[15] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Annual International Cryptology Conference*, pp. 410–424, Springer, 1997.

[16] K. Sampigethaya and R. Poovendran, "A survey on mix networks and their secure applications," *Proceedings of the IEEE*, vol. 94, no. 12, pp. 2142–2181, 2006.

[17] P. S. Barreto, H. Y. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," in *Annual international cryptology conference*, pp. 354–369, Springer, 2002.

[18] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Cryptographers' Track at the RSA Conference*, pp. 111–126, Springer, 2016.

[19] "Verifiable credentials data model v1.1: Expressing verifiable information on the web," Nov 2021.

[20] S. Yonezawa, T. Kobayashi, and T. Saito, "Pairing-friendly curves," *Network Working Group. Internet-Draft. January*, 2019.

[21] NymTech, "Coconutgo." https://github.com/nymtech/coconut, 2021.

[22] Auth0, "User profile structure."

[23] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Annual international cryptology conference*, pp. 56–72, Springer, 2004.

[24] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials," *Cryptology ePrint Archive*, 2013.

[25] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1348–1366, IEEE, 2021.

[26] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for tls," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1919–1938, 2020.