PAPR: Publicly Auditable Privacy Revocation for Anonymous Credentials

Joakim Brorsson[1⋆], Bernardo David[2⋆⋆], Lorenzo Gentile[2⋆⋆⋆], Elena Pagnin[3], and Paul Stankovski Wagner[1†]

[1] Lund University, Lund, Sweden
joakim.brorsson@eit.lth.se,paul.stankovski_wagner@eit.lth.se
[2] IT University of Copenhagen, Copenhagen, Denmark
bernardo@bmdavid.com,lorg@itu.dk
[3] Chalmers University of Technology, Gothenburg, Sweden
elenap@chalmers.se

Abstract. We study the notion of anonymous credentials with *Publicly Auditable Privacy Revocation* (PAPR). PAPR credentials simultaneously provide *conditional* user privacy and *auditable* privacy revocation. The first property implies that users keep their identity private when authenticating unless and until an appointed authority requests to revoke this privacy, retroactively. The second property enforces that auditors can verify whether or not this authority has revoked privacy from an issued credential (*i.e.* learned the identity of the user who owns that credential), holding the authority accountable. In other words, the second property enriches conditionally anonymous credential systems with transparency by design, effectively discouraging such systems from being used for mass surveillance. In this work, we introduce the notion of a PAPR anonymous credential scheme, formalize it as an ideal functionality, and present constructions that are provably secure under standard assumptions in the Universal Composability framework. The core tool in our PAPR construction is a mechanism for randomly selecting an anonymous committee which users secret share their identity information towards, while hiding the identities of the committee members from the authority. As a consequence, in order to initiate the revocation process for a given credential, the authority is forced to post a request on a public bulletin board used as a broadcast channel to contact the anonymous committee that holds the keys needed to decrypt the identity connected to the credential. This mechanism makes the user de-anonymization publicly auditable.

⋆ This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation

⋆⋆ This work was supported by the Concordium Foundation and by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B.

⋆⋆⋆ This work was supported by the Concordium Foundation.

† This work was supported by was supported by the Swedish Foundation for Strategic Research, grant RIT17-0035.

1 Introduction

Ensuring user privacy while complying with requirements for user accountability is often a challenging task. As an example, consider an on-line payment platform. User privacy demands that identities remain unknown while performing on-line payments, while Know Your Customer and Anti-Money Laundering regulations demand that misbehaving users should be held accountable. This and many more sophisticated examples motivate the analysis of the trade-offs between user privacy and accountability, both from a technical perspective [18, 19, 36, 40, 56], and from an ethical standpoint [1, 45, 59].

The notion of conditional privacy captures settings where a set of authorities is given the power to revoke a user's privacy. Unfortunately, the vast majority of existing systems that provide conditional privacy naïvely trust revocation authorities to trigger privacy revocation only when a user behaves suspiciously. Thus, they do not hold authorities accountable, allowing them to surreptitiously revoke privacy. In particular, third party auditors (*e.g.* regulatory agencies and users themselves) cannot verify whether privacy revocation has happened (or not). As a consequence, user trust in the privacy of such systems is eroded.

We address this issue by introducing the notion of Publicly Auditable Privacy Revocation (PAPR). In schemes offering conditional privacy, PAPR makes the actions of authorities transparent to third party auditors, who can monitor when privacy revocation takes place and thus detect abuse of power by the authorities. We showcase the power (and challenges) of this notion by showing how to add PAPR to anonymous credential schemes in order to achieve increased (user) trust via strong accountability guarantees for both users and authorities.

1.1 Related Works

Privacy Preserving Authentication allows users to authenticate without revealing their true identities. This feature is crucial for systems with strong user privacy requirements, and can be achieved in many ways. Anonymous credentials, envisioned by Chaum in [27] and first realized with provably security in [20], allow users to prove ownership of a valid credential without revealing their identity. Later, anonymous credential schemes with improved efficiency [21, 9, 7] were proposed. Schemes with richer features such as delegation [30] and attributes [21, 7, 12] have also been proposed. More recently, universally composable [22] anonymous credentials were proposed in [16, 15]. In anonymous credential schemes, there are two main strategies to prevent abuse of anonymity: allow users to authenticate anonymously only a predetermined number of times [58, 17]; or introduce mechanisms for privacy revocation by a central authority [20].

Conditional Privacy (or *revocable privacy* [56]) combines user anonymity and accountability, so that it is possible for an authority to revoke a user's right to privacy, should the target user behave in illicit ways. This is often implemented by giving a selected group of trusted entities the power to revoke confidentiality or anonymity guarantees as needed. In order to avoid malicious strategies, there

is an unwillingness by authorities to let users decide who these trusted parties should be. Instead, a set of central *privacy revocation authorities* is often used. This is the case in many applications, including encryption systems [53], e-cash [14], blind signatures [57] and group signatures [28].

Public Auditability was introduced as a way to make authorities accountable for their actions and thereby prevent abuse of power. Techniques for public auditability are often application specific. Examples include auditing the behaviour of pseudonym conversion authorities [19] or auditing that certificate authorities provide correct public keys [48, 52]. Known approaches to obtain auditability for privacy revocation authorities in the context of anonymous credentials either use non-standard techniques, such as witness encryption [42], or rely on a set of trusted authorities that are assumed not to collude [14, 28, 50, 53, 57].

Anonymous Committees address the problem of ensuring that a set of parties do not collude, by establishing a committee where the members' identities are not known to any party, including the committee members themselves (*i.e.* a member knows it is in the committee but does not know the identity of other members). Several works exist on this problem, e.g. [29, 46, 32, 31]. In this setting, it is both hard for committee members to collude and for an adversary to subvert committee members.

In particular, the idea of distributing sensitive information to anonymous committees (*e.g.* privacy revocation trapdoors) or having anonymous committees execute cryptographic protocols has been explored in the context of proactive secret sharing [11, 41, 26], multiparty computation (MPC) [38] and threshold encryption [35]. These protocols work in the so called You Only Speak Once (YOSO) model, where a fresh randomly chosen anonymous committee executes each round of the protocol, limiting the adversary to probabilistic corruptions (*i.e.* when the adversary corrupts any party, it only knows that this party may be party of the current committee with a certain probability smaller than 1).

Concurrent Work which addresses a similar goal of authority accountability was proposed in [34]. However, this scheme does not achieve any notion of composability and cannot be easily proven UC secure. Moreover, the committee that is expected to cooperate in order to revoke privacy is not hidden, so its publicly known members may be corrupted by a proactive adversary.

1.2 Our Contributions

We introduce the concept of anonymous credentials with PAPR, which we model and construct in the Universal Composability [22] framework. We define this new concept as an ideal functionality supporting standard actions of anonymous credentials issuance, linkable[4] credential showing and privacy revocation. Our

[4] While many anonymous credential schemes strive to provide unlinkability among different showings, we restrict ourselves to the simpler case where different showings of the same credential can be linked in order to focus on our new PAPR techniques.

ideal functionality captures the novel PAPR property by guaranteeing that all parties are notified when the issuer performs privacy revocation on a credential. Enforcing this guarantee is the main challenge in our construction.

The core of our contribution is a novel mechanism to distributively store the secret identity connected to a user's anonymous credential in such a way that privacy revocation is possible, but any attempt to revoke privacy (by retrieving the user's identity) requires a public announcement of the privacy revocation act of the corresponding credential. Our contributions are summarized as follows:

- We introduce the notion of Publicly Auditable Privacy Revocation (PAPR) for anonymous credential schemes.
- We provide a security definition of anonymous credentials with PAPR in the Universal Composability framework (Section 3).
- We construct an efficient anonymous credential scheme that achieves our PAPR notion with UC security against static malicious adversaries under standard assumptions (Section 4).
- We show how to modify our construction to obtain a PAPR anonymous credential scheme that is UC-secure against mobile adversaries via proactive secret sharing and threshold encryption in the YOSO model (Section 5).

1.3 Overview of our Techniques

At a high level, our approach to create an anonymous credential scheme with publicly accountable privacy revocation can be summarized in the following three steps. First, the system maintains one global public list of enrolled parties $\mathcal{P}$ (committee candidates), consisting of party identifiers $ID_{\mathcal{P}}$, e.g., a name, and identity keys $\mathsf{pk}_{\mathcal{P}}$ (leveraging a PKI). Second, the issuer produces credentials for a user, only if: *(a)* the user proves to have shared their identity key to an *anonymous* committee, *(b)* the committee is composed by a fixed number of *other parties* in the system (*i.e.* from the committee candidates), *(c)* the selection of committee parties was *provably at random*. Third, any credential can be subject to privacy revocation upon public announcement. The goal of privacy revocation is to let an authority identify the holder of a given *anonymous* credential pk_C. Concretely, this is achieved by obtaining the credential holder's *identity key* $\mathsf{pk}_{\mathcal{P}}$ which is linked to the party's identity $ID_{\mathcal{P}}$ via a public key infrastructure.

We remark that with this approach, no data is actually sent by the user to the anonymous committee members during credential issuance. Instead the data is stored within a bulletin board. The bulletin board is also used to publicly announce privacy revocation since the identities of the (anonymous) committee members are hidden from the issuer. The main challenge we face in PAPR is to simultaneously hide the identity of committee members and guarantee the random selection of the committee.

The Main Protocol The core idea in our main construction of PAPR anonymous credentials is to enable users to sample a random and anonymous committee in a verifiable way, using a verifiable shuffle. The protocol leverages a Public Key

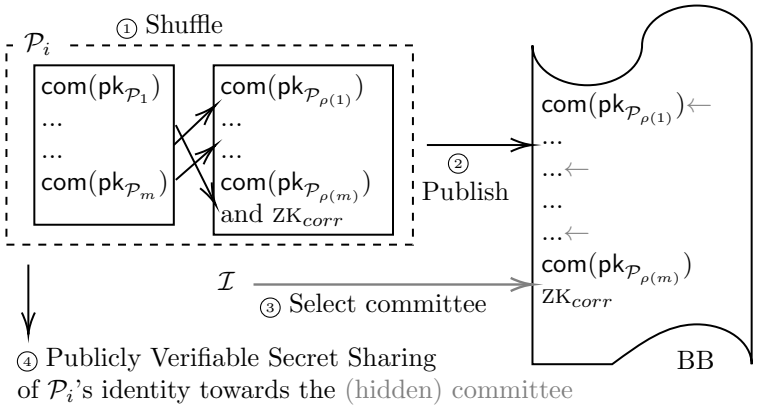

Fig. 1: Mechanics of $\prod_{PC}$: ① Each user $\mathcal{P}_i$ locally generates commitments to hide each committee candidate's public key. Then, the party shuffles the set of commitments in a provable way (ZK_{corr}). ② The output of the shuffle is published on a public bulletin board (BB) by $\mathcal{P}_i$. ③ The issuer $\mathcal{I}$ selects the committee members for $\mathcal{P}_i$ from the shuffled list. ④ $\mathcal{P}_i$ secret shares its identity towards the selected committee members in a publicly verifiable way.

Infrastructure where keys for all m users are registered. Intuitively, to establish an anonymous committee, a user commits to all user public keys in the list, shuffles (*i.e.* permutes and re-randomizes) the initial commitments and proves that it has done so correctly, posting the resulting commitments and proof to a Public Bulletin Board (BB). The issuer then selects the committee from the shuffled commitments by publishing $n < m$ random indices on the BB. This approach to committee selection is illustrated in Figure 1.

A credential request requires the user to publish secret shares of its identity encrypted under the public key of the selected committee along with zero knowledge proofs of share validity (*i.e.* providing a publicly verifiable secret sharing of its identity). This creates a link between the credential and the encrypted shares of the identity, without revealing which identity was shared.

Since the issuer cannot learn the identity of the members of the privacy revocation committee, it can only trigger privacy revocation for any issued credential by posting a public request on the BB. The committee members, monitoring the BB, reacts to such a request and proceed to reconstruct the user's identity by providing the decrypted shares to the issuer via a private channel.

We stress that both during committee establishment and secret sharing to the committee, all computation and communication is carried out by the user and the issuer only, without involving the committee members at all.

In this protocol, differently from the YOSO model, we allow the party who requests a credential to learn the identities of the corresponding committee members. The rationale is that, as far as static security is concerned, an adversary playing as a malicious user can already link the identity of a corrupted committee member to an anonymous credential. Letting the identities of the elected

committee members be known to the requesting party in this way thus creates no incentive of corruption, as it leaks no additional information. We stress that while the identities of committee members are learned, the selecting party still has no influence over what parties constitute the committee since they are selected provably at random.

Proactively Secure Versions Our main protocol is only secure against static adversaries. To withstand mobile adversaries, who can periodically uncorrupt parties and corrupt new parties, a heavier machinery is needed. It is crucial to notice that mobile adversaries in our setting can 1) corrupt a majority of the committee that holds revocation data for a corrupted party's credential, which would allow an adversary to block privacy revocations, and 2) gradually corrupt a majority of the committee holding revocation data for an honest party (by moving to a new disjoint set of parties every epoch), which would allow it to stealthily learn the honest party's identity. Such mobile adversaries could be trivially addressed by computing the steps for issuing and revoking a credential via YOSO MPC, where each round of the computation is performed by a fresh randomly chosen fully anonymous committee, preventing the adversary from corrupting the committee currently holding the computation's secret state. However, YOSO MPC is notoriously expensive. Therefore, as a first step towards security against a mobile adversary, we instead show that we can use proactive secret sharing in the YOSO model, where committees are not known to *any* party, and the shared revocation data is periodically transferred to a new randomly chosen anonymous committee. While this technique solves the issue in a simple way, it requires the YOSO committees to hold an amount of data linear in the number of credentials issued.

An even more efficient alternative for proactive security is to employ YOSO threshold encryption and adding distributed key generation to our setup phase to obtain a system wide public encryption key. Issuance is then modified so that each party publishes an encryption of its identity under this common encryption key and proves in zero knowledge that they have done so in a way that creates a link between this encryption and the issued credential. Revocation can then be done by threshold-decrypting the ciphertext connected to that credential. The advantages of the latter approach are twofold, it both makes credential issuance simpler for parties (*i.e.* they generate one ciphertext instead of encrypting multiple shares), and improves communication complexity for the YOSO committee members, since they only have to hold shares of a single secret key.

2 Preliminaries

Throughout the paper $\lambda \in \mathbb{N}$ denotes a security parameter. We will use the notation $\vec{a}[i]$ to denote the i'th element of the vector $\vec{a}$. Finally, when signing messages not in the message space of the signature algorithm (*e.g.* a group element or a vector), we let the conversion to the message space be implicit.

2.1 Cryptographic Primitives

Our construction employs a key-private encryption scheme (*i.e.* an encryption which hides the recipient's public key) $\mathsf{Enc} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, a signature scheme $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, a commitment scheme $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$, and Shamir Secret Sharing [55]. Details on these schemes are presented in Appendix A.

We further use two special types of digital signature schemes, structure preserving signatures (SPSig) [3], and blind signatures (BSig) [54]. Structure preserving signatures are digital signatures where signatures σ and messages m belong to the same space. Blind signatures are a variant of signatures where the signer does not learn the message she signs. In known constructions the blind signature generation procedure is an interactive protocol between the signer and the party wishing to have a message signed.

We use a non-interactive zero-knowledge (NIZK) proof of shuffle correctness for commitments defined as the triple of algorithms $\mathsf{Shuf} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as per Definition 1. This NIZK allows for proving that a certain (public) vector of commitments was obtained by re-randomizing a given (public) vector of commitments and permuting the re-randomized commitments without revealing the randomness used for re-randomization nor the permutation. This NIZK can be efficiently realized from the proof of shuffle correctness for ciphertexts of [8]. In our setting, we view an ElGamal ciphertext as a commitment and use proofs of commitment shuffle correctness to convince a verifier that two distinct sets of commitments yield the same set of openings. The definitions of completeness, soundness and zero-knowledge for Shuf follow the same structure and aims as in [8] and are available in Appendix A.

Definition 1 (Provable Shuffle of Commitments). *A proof system* $\mathsf{Shuf} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for proving shuffle of commitments generated by a commitment scheme* C *consists of the following algorithms.*

$\mathsf{Shuf.Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs public parameters* pp*, often referred to as the common reference string (implicitly input to all subsequent algorithms).*

$\mathsf{Shuf.Prove}(n, \rho, \{c_i\}_{i\in[n]}) \rightarrow (\{c'_i\}_{i\in[n]}, \pi)$*: The provable shuffle algorithm takes as input an integer* n*, a permutation* ρ *over the set* $\{1, \ldots, n\}$*, and* n *commitments* $\{c_i\}_{i\in[n]}$ *generated by* C.Commit*. It returns a list of* n *commitments* $\{c'_i\}_{i\in[n]}$ *and a proof* π*.*

$\mathsf{Shuf.Verify}(n, \{c_i\}_{i\in[n]}, \{c'_i\}_{i\in[n]}, \pi) \rightarrow v$*: The verification algorithm takes as input an integer* n*, two sets of* n *commitments and a proof* π*. It returns 1 (accept) if* π *is a valid proof for the relation "there exists a set* $M = \{m_i\}_{i\in[n]}$ *and a permutation* $\rho \in S_n$ *s.t.* $\{\mathsf{C.Open}(c_i, m_i, r_i)\}_{i\in[n]} = \{\mathsf{C.Open}(c'_{\rho(i)}, m_{\rho(i)}, r'_{\rho_i}\}_{i\in[n]}$*", where the randomnesses* r_i, r'_i *are extracted from* π*. Otherwise it returns 0 (reject).*

2.2 Universal Composability and Ideal Functionalities

In the Universal Composability (UC) framework [22] the security of a protocol is analyzed under the real-world/ideal-world paradigm, *i.e.*, by comparing the real world execution of a protocol with an ideal world interaction with the ideal functionality that it realizes. Protocols that are secure in the UC framework can be arbitrarily composed with each other without compromising security. In the ideal world execution, dummy parties (potentially controlled by an ideal adversary $\mathcal{S}$, referred to as the simulator) interact with an ideal functionality $\mathcal{F}$. In the real world execution, parties (potentially corrupted by a real world adversary $\mathcal{A}$) interact with each other by following a protocol π that realizes the ideal functionality $\mathcal{F}$. The real and ideal executions are controlled by the environment $\mathcal{Z}$, an entity that controls inputs and reads the outputs of each party, $\mathcal{A}$ and $\mathcal{S}$. The protocol π securely realizes $\mathcal{F}$ in the UC framework if the environment $\mathcal{Z}$ cannot efficiently distinguish between the real world execution with π and $\mathcal{A}$ and the ideal world execution with $\mathcal{S}$ and $\mathcal{F}$.

Specifically we make use of a set of ideal functionalities $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$. These functionalities are described in detail in Appendix B, we here only give an overview of them. Briefly, the bulletin board functionality $\mathcal{F}_{BB}$, works so that any party can publish a message m to the board by sending (POST, sid, m) and read the contents of the board by sending (READ, sid). $\mathcal{F}_{PKI}$ is a functionality where each party can only send (POST, sid, m) once and can retrieve party $\mathcal{P}$'s message as $(\text{READ}, sid, \mathcal{P})$. The functionality for interactive zero knowledge, $\mathcal{F}_{ZK}$ is defined so that a prover $\mathcal{P}$ can send $(\text{ZK-PROVER}, sid, \mathcal{V}, x, w)$ to $\mathcal{F}_{ZK}$, which sends $(\text{ZK-PROOF}, sid, x)$ to the verifier $\mathcal{V}$ only if w is a witness for the statement x. Analogously, the functionality for non-interactive zero knowledge $\mathcal{F}_{NIZK}$ is defined by $(\text{PROVE}, sid, x, w)$, returning a proof π guaranteeing that w is a witness for the statement x, and $(\text{VERIFY}, sid, x, \pi)$, outputting 1 for a valid π for the statement x.

3 Defining PAPR for Anonymous Credentials

In this section we introduce the notion of a Publicly Auditable Privacy Revocation (PAPR) Anonymous Credential Scheme and describe an ideal functionality $\mathcal{F}_{PC}$ for it. Section 4 presents our protocol Π_{PC} that realizes $\mathcal{F}_{PC}$ based on efficient and well-known building blocks. Section 4.1 proves Π_{PC} secure in the presence of a static, malicious adversary in the UC framework [22].

Defining PAPR Credentials We define the notion of PAPR credentials as the ideal functionality $\mathcal{F}_{PC}$ presented in Figure 3. This functionality provides standard anonymous credential interfaces supporting requesting credentials (CRED-REQ), issuing credentials (ISSUE-CRED), and showing credentials (SHOW-CRED). While any party may request a credential, only a special party called the *issuer* may approve such a request. As usual, requesting an anonymous credential and later showing it does not reveal any information about the credential owner's identity to the issuer nor to the party who is shown a credential. However, we do

not aim at achieving unlinkability across multiple credential showings. In order to capture the novel PAPR property, the identity revocation interface (ANNOUNCE-REV) allows the issuer to request the identity of the owner of a given credential at any time, but this also immediately informs all other parties that privacy has been revoked for that credential.

Functionality $\mathcal{F}_{PC}$

$\mathcal{F}_{PC}$ is parameterized by a credential space $\mathcal{PK}$. The functionality interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$, a special party called the issuer $\mathcal{I} = \mathcal{P}_{m+1}$ and the ideal adversary $\mathcal{S}$. It keeps a list L_{cred} of credentials and a setup list L_{setup}, both initialized to $\emptyset$.

Setup: On input (SETUP, *sid*) from $\mathcal{P}_i$, add that party to the list L_{setup}.

Credential Request: On input (CRED-REQ, *sid*) from $\mathcal{P}_i$, if $L_{setup} \neq \mathcal{P} \cup \mathcal{I}$, then ignore the request. If $\mathcal{P}_i$ is honest, sample a random pk_{C_i} from $\mathcal{PK}$ and send (CRED-REQ, *sid*) to $\mathcal{S}$. Otherwise send (KEY?, *sid*) to $\mathcal{S}$ and await response (KEY, *sid*, pk_{C_i}). Finally write $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 0)$ to L_{cred} and send (CRED-REQ, *sid*, pk_{C_i}) to $\mathcal{I}$.

Credential Issuance: On input (ISSUE-CRED, *sid*, pk_{C_i}) from $\mathcal{I}$, if $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 0) \in L_{cred}$, update the entry to $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 1)$ and send (CRED, *sid*, pk_{C_i}) to $\mathcal{P}_i$ and $\mathcal{S}$. Else write $(\perp, \mathsf{pk}_{C_i}, 1)$ to L_{cred}.

Credential Showing: On input (SHOW-CRED, *sid*, $\mathsf{pk}_{C_i}, \mathcal{P}_j$) from $\mathcal{P}_i$, if $(\cdot, \mathsf{pk}_{C_i}, 1) \notin L_{cred}$, ignore the request. Send (VALID-CRED, *sid*, pk_{C_i}) to $\mathcal{P}_j$ and (VALID-CRED, *sid*, $\mathsf{pk}_{C_i}, \mathcal{P}_j$) to $\mathcal{S}$.

Privacy Revocation: On input (ANNOUNCE-REV, *sid*, pk_{C_i}) by $\mathcal{I}$, send (ANNOUNCE-REV, *sid*, pk_{C_i}) to all $\mathcal{P}_j \in \mathcal{P}$ and $\mathcal{S}$. If $(\cdot, \mathsf{pk}_{C_i}, 1) \notin L_{cred}$, then ignore the request. If $(\perp, \mathsf{pk}_{C_i}, 1) \in L_{cred}$, then delete $(\perp, \mathsf{pk}_{C_i}, 1)$ from L_{cred}, and ignore the request. Else, $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 1) \in L_{cred}$, then delete $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 1)$ from L_{cred}, output (IDENTITY, *sid*, $\mathcal{P}_i, \mathsf{pk}_{C_i}$) to $\mathcal{S}$ and send a delayed output (IDENTITY, *sid*, $\mathcal{P}_i, \mathsf{pk}_{C_i}$) to $\mathcal{I}$.

Fig. 2: Ideal functionality $\mathcal{F}_{PC}$ for PAPR Credentials.

4 Realizing PAPR for Anonymous Credentials

In Figures 3 and 5 we describe protocol $\prod_{PC}$ for anonymous credentials with PAPR. We consider *malicious* adversaries that may deviate from the protocol in any arbitrary way. Moreover, in this section we consider the *static* case, where the adversary is only allowed to corrupt parties before protocol execution starts and parties remain corrupted (or not) throughout the execution. We assume that parties have access to synchronous communication channels, *i.e.*, all messages are delivered with a known maximum delay. To be concise, in the protocol description we let all reads from $\mathcal{F}_{BB}$ and $\mathcal{F}_{PKI}$ be implicit. It is also implicit that

if a variable that is part of a procedure (*e.g.*, a public key) is not yet available on $\mathcal{F}_{PKI}$ or $\mathcal{F}_{BB}$, the current procedure will terminate without output (*i.e.*, ignore the procedure call). Lastly, to avoid undefined behaviour while keeping the protocol description simple, whenever more than one valid message with equal values exist on $\mathcal{F}_{BB}$, only the chronologically first message shall be considered. We further assume that a user remains anonymous when posting to $\mathcal{F}_{BB}$ as is the case in the YOSO model.

Using Committees We assume that committees are formed by selecting uniformly at random the smallest number n of parties from set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ such that every committee is guaranteed an *honest majority* with overwhelming probability given a certain corruption ratio. Selecting committees in this way has been explored extensively in [33], where concrete numerical examples of its size are provided. Indeed, a few examples are available in Section 6.

Since all parties are potential committee members, they are expected to monitor the bulletin board. Notice, however, that our protocol works with privacy revocation committees selected from any set of parties (potentially disjoint from the set of parties who request credentials, as discussed in Section 6.2) as long as these committees have honest majority with overwhelming probability.

Protocol Overview We now give a step-by-step overview of protocol $\prod_{PC}$.

Setup The **Issuer Setup** and **User Setup** procedures consist of enrolling keys for the parties in the system. Note that, by registering its identity key $\mathsf{pk}_{\mathcal{P}_i}$ to the PKI, the user key and identity are linked. This link forms the basis for user identification during privacy revocation.

Before a credential can be issued, a committee with which each party's identity key will be shared must be established. Each party first executes the **Hide Committee Candidates** procedure. In step (a) the party hides the order of the committee candidates using a verifiably random shuffle, and is then (anonymously) bound to the shuffle by signing it with sk_T. In step (b), it publishes the shuffle, proof, and signature on the bulletin board.

The issuer then in step (a) of the **Sample Committee** procedure verifies that the requesting party has published a single signed and valid shuffle. If so, in step (b) it responds with a set of random indexes, indicating which of the shuffled values in $\vec{f}'$ shall constitute the committee.

Credential Issuance In the **Credential Request** procedure, a user in step (1) collects the public keys of the committee as indicated by $\mathcal{I}$ into $\vec{h}_i$, It also puts the corresponding commitments to the committee keys into $\vec{c}_i$. It then in step (2) produces a vector of encrypted shares $\vec{E}_i$ of its enrolled identity public key $\mathsf{pk}_{\mathcal{P}_i}$ for the committee in $\vec{h}_i$. To allow other users to know whether they are in the committee, a set of indicators, $\vec{q}_i$, is also produced. A party knows it is the j'th member of a committee if $\vec{q}_i[j]$ decrypts to its public key. Before generating credential keys in step (4) and posting the credential request in step (5), a party

Protocol $\prod_{PC}$ (First Part)

Protocol $\prod_{PC}$ is executed by an issuer $\mathcal{I}$ and parties $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ interacting with functionalities $\mathcal{F}_{PKI}$, $\mathcal{F}_{BB}$, $\mathcal{F}_{NIZK}$ and $\mathcal{F}_{ZK}$. $\prod_{PC}$ is parameterized by a constant $n \in \mathbb{Z}$ such that sampling n parties out of $\{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ yields an honest majority except with negligible probabilty.

Setup: The issuer $\mathcal{I}$ and all parties $\mathcal{P}_i$ proceed as follows:

1. (Issuer Setup) On input (SETUP, sid), $\mathcal{I}$ generates a blind signature keypair $(\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ with $\lambda \in \mathbb{N}$ being a security parameter, an enrollment keypair $(\mathsf{pk}_E, \mathsf{sk}_E) \leftarrow \mathsf{SPSig.KeyGen}(\mathsf{pp})$, a revocation keypair $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and an issuance keypair $(\mathsf{pk}_I, \mathsf{sk}_I) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and sends $(\text{REPORT}, sid, (\mathsf{pk}_B, \mathsf{pk}_E, \mathsf{pk}_R, \mathsf{pk}_I))$ to $\mathcal{F}_{PKI}$.
2. (User Setup) On input (SETUP, sid), $\mathcal{P}_i$ generates user identity keys $(\mathsf{pk}_{\mathcal{P}_i}, \mathsf{sk}_{\mathcal{P}_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and sends $(\text{REPORT}, sid, \mathsf{pk}_{\mathcal{P}_i})$ to $\mathcal{F}_{PKI}$. Additionally $\mathcal{P}_i$ generates a single-use token keypair $(\mathsf{pk}_{T_i}, \mathsf{sk}_{T_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and interacts with $\mathcal{I}$ over a secure channel to obtain signatures $\sigma_E(\mathsf{pk}_{\mathcal{P}_i}) \leftarrow \mathsf{SPSig.Sign}(\mathsf{sk}_E, \mathsf{pk}_{\mathcal{P}_i})$. Finally $\mathcal{P}_i$ runs $\mathsf{BSig.User}(\mathsf{pk}_B, \mathsf{pk}_{T_i})$ with $\mathcal{I}$ running $\mathsf{BSig.Sign}(\mathsf{sk}_B)$ so as to compute the blind signature $\sigma_B(\mathsf{pk}_{T_i})$.
3. (Hide Committee Candidates) Let $\vec{\mathsf{pk}}_{\mathcal{P}}$ be the vector of all $\mathsf{pk}_{\mathcal{P}_j}$ and $\vec{f}$ the vector, s.t. $\vec{f}[j] = \mathsf{C.Commit}(\vec{\mathsf{pk}}_{\mathcal{P}}[j], 1)$, then $\mathcal{P}_i$ proceeds as follows:
   (a) Sample a random permutation ρ_i and verifiably shuffle $\vec{f}$ as $(\vec{f}'_i, \pi_{\rho_i}) = \mathsf{Shuf.Prove}(m, \rho_i, \vec{f})$. Sign the shuffle as $\sigma_{T_i}(\vec{f}'_i) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{T_i}, \vec{f}'_i)$,
   (b) Send $(\text{POST}, sid, (\texttt{hide}, \vec{f}'_i, \pi_{\rho_i}, \mathsf{pk}_{T_i}, \sigma_{T_i}(\vec{f}'_i), \sigma_B(\mathsf{pk}_{T_i}))$ to $\mathcal{F}_{BB}$.
4. (Sample Committee) When $(\text{POST}, sid, (\texttt{hide}, \vec{f}'_j, \pi_{\rho_j}, \mathsf{pk}_{T_j}, \sigma_{T_j}(\vec{f}'_j), \sigma_B(\mathsf{pk}_{T_j}))$ appers on $\mathcal{F}_{BB}$, $\mathcal{I}$ proceeds as follows:
   (a) Check that $1 = \mathsf{BSig.Verify}(\mathsf{pk}_B, \mathsf{pk}_{T_j}, \sigma_B(\mathsf{pk}_{T_j}))$, $1 = \mathsf{Sig.Verify}(\sigma_{T_j}(\vec{f}'_j))$ and $1 = \mathsf{Shuf.Verify}(m, \vec{f}, \vec{f}'_j, \pi_{\rho_j})$.
   (b) If true, let $\vec{b}_j$ where $\vec{b}_j[i] \xleftarrow{\$} \mathbb{Z}^*_{m+1}, |\vec{b}_j| = n$, indicate the indexes selected for the committee, sign $\vec{b}_j$ as $\sigma_I(\vec{b}_j) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_I, \vec{b}_j)$, send $(\text{POST}, sid, (\texttt{sample}, \vec{f}'_j, \vec{b}_j, \sigma_I(\vec{f}'_j || \vec{b}_j)))$ to $\mathcal{F}_{BB}$ and store $(\vec{f}'_j, \vec{b}_j)$ internally.

Credential Request: On input $(\text{CRED-REQ}, sid)$, if there is an entry $(\texttt{sample}, \vec{f}'_j, \vec{b}_j, \sigma_I(\vec{f}'_j || \vec{b}_j))$ where $1 = \mathsf{Sig.Verify}(\mathsf{pk}_I, \sigma_I(\vec{f}'_j || \vec{b}_j)$ on $\mathcal{F}_{BB}$, $\mathcal{P}_i$ proceeds as follows:

1. Define $\vec{a}_i[j] = \vec{\mathsf{pk}}_{\mathcal{P}}[\rho(j)]$, $\vec{h}_i[j] = \vec{a}_i[\vec{b}_i[j]]$ and $\vec{c}_i[j] = \vec{f}'_i[\vec{b}_i[j]]$ for $j = 1, \ldots, n$.
2. Generate identity shares via Shamir secret sharing, *i.e.* sample a random polynomial $f()$ of degree $\lceil \frac{n}{2} \rceil$ where $f(0) = \mathsf{pk}_{\mathcal{P}_i}$ and set $\vec{s}_i[j] = f(j)$ for $j = 1, \ldots, n$. Encrypt the shares under the committee public keys obtaining $\vec{E}_i[j] = \mathsf{Enc.Encrypt}(s_i[j], \vec{h}_i[j])$ and construct committee member indicators $\vec{q}_i[j] \leftarrow \mathsf{Enc.Encrypt}(\vec{h}_i[j], \vec{h}_i[j])$ for $j = 1, \ldots, n$.
3. Prove correct escrow by sending $(\text{PROVE}, sid, x_i, w)$ to $\mathcal{F}_{NIZK}$ and getting $(\text{PROOF}, sid, \pi_{esc_i})$, where x_i and w are defined as in ZK_{esc} (Figure 4).
4. Generate user credential keys $(\mathsf{pk}_{C_i}, \mathsf{sk}_{C_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$.
5. Sends $(\text{POST}, sid, (\texttt{req}, \vec{E}_i, \vec{f}'_i, \vec{q}_i, \mathsf{pk}_{C_i}, x_i, \pi_{esc_i}))$ to $\mathcal{F}_{BB}$.

Fig. 3: $\prod_{PC}$ - Setup, Committee Establishment and Credential Request.

must first prove correct sharing in step (3). We provide a detailed description of the proven relation ZK_{esc} in the next subsection below.

When the issuer observes a credential request on the bulletin board it first executes step (1) of the **Credential Issuance** procedure to verify that a committee has been formed. Step (2) is executed to verify that sharing is done correctly by the requesting user. If all checks pass, step (3) is executed to sign the credential and publish it.

$$\text{ZK}_{esc}\{\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}, \sigma_E(\mathsf{pk}_{\mathcal{P}}), \vec{h}, \vec{s}, \vec{r} \mid \text{ZK}_{ID} \wedge \text{ZK}_{share}\}$$

$$\begin{array}{ll} \textcircled{1}\ \text{ZK}_{ID}\{\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}, \sigma_E(\mathsf{pk}_{\mathcal{P}}) \mid & \textcircled{2}\ \ \text{ZK}_{share}\{\vec{h}, \vec{s}, \mathsf{pk}_{\mathcal{P}} \mid \\ \mathsf{Sig.VerifyKey}(\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}) \wedge & \textcircled{2.1}\ \mathsf{pk}_{\mathcal{P}} = \mathsf{SShare.Reconstruct}(\vec{s}) \wedge \\ \mathsf{SPSig.Verify}(\mathsf{pk}_E, \mathsf{pk}_{\mathcal{P}}, \sigma_E(\mathsf{pk}_{\mathcal{P}}))\} & \textcircled{2.2}\ \forall j \in \{1, \ldots, n\} : \\ & \textcircled{2.3}\ \vec{E}[j] = \mathsf{Enc.Encrypt}(\vec{s}[j], \vec{h}[j]) \wedge \\ & \textcircled{2.4}\ \mathsf{C.Open}(\vec{c}[j], \vec{h}[j], \vec{r}[j]) \wedge \\ & \textcircled{2.5}\ \vec{q}[j] = \mathsf{Enc.Encrypt}(\vec{h}[j], \vec{h}[j])\ \} \end{array}$$

Fig. 4: Elements of the ZK_{esc} statement. Intuitively, ZK_{ID} states that the proving user controls the enrolled identity key $\mathsf{pk}_{\mathcal{P}}$. ZK_{share} states that the identity key pk_U has been correctly shared to the committee members in $\vec{h}$.

Proving Correct Escrow The correctness of the identity escrow in a credential request is defined by the relation ZK_{esc}. Figure 4 defines ZK_{esc} on a high level, *i.e.* by using procedure definitions. To simplify notation, we here define a procedure for knowledge of a private key, $\mathsf{Sig.VerifyKey}(\mathsf{sk}, \mathsf{pk}) \rightarrow v$, which indicates if sk, pk is a valid keypair with respect to $\mathsf{Sig.KeyGen}(.)$.

For illustrative purposes, we define ZK_{esc} as a conjunction, where $\text{ZK}_{esc} = \{\text{ZK}_{ID} \wedge \text{ZK}_{share}\}$. The first part, $\textcircled{1}\ \text{ZK}_{ID}$, states that the prover is the owner of $\mathsf{pk}_{\mathcal{P}}$, *i.e* it knows secret key $\mathsf{sk}_{\mathcal{P}}$, and an issuer signature, $\sigma_E(\mathsf{pk}_{\mathcal{P}})$, on $\mathsf{pk}_{\mathcal{P}}$. The second part, $\textcircled{2}\ \text{ZK}_{share}$ is a statement that $\textcircled{2.1}$ the shares are constructed correctly, *i.e.* any set of k shares will reconstruct to the users public key $\mathsf{pk}_{\mathcal{P}}$. Further, $\textcircled{2.2}$ each of these shares, $\textcircled{2.3}$ is correctly encrypted, $\textcircled{2.4}$ for the correct committee member, $\textcircled{2.5}$ which is correctly indicated in $\vec{q}$.

Credential Showing The **Credential Showing** and **Verify Credential Showing** procedures are straightforward zero knowledge proofs of knowledge of the credential private key sk_{C_i} for the public key pk_{C_i} (and when verifying, also checking that the shown credential has been issued by $\mathcal{I}$ and that the credential is not revoked).

Privacy Revocation To learn the secret identity behind a credential public key pk_{C_j}, *i.e.* to revoke the privacy, the issuer (and only the issuer) can execute

Protocol $\prod_{PC}$ (Second Part)

Credential Issuance: On input (ISSUE-CRED, sid, pk_i), if there is an entry $(\mathtt{req}, \vec{E}_j, \vec{f}'_j, \vec{q}_j, \mathsf{pk}_{C_j}, x_i, \pi_{esc_i})$ on $\mathcal{F}_{BB}$, $\mathcal{I}$ does the following:

1. If no internal entry $(\vec{f}'_j, \vec{b}_j)$ exists, ignore the next steps.
2. Send (VERIFY, sid, x_j, π_{esc_j}) to $\mathcal{F}_{NIZK}$, await the reply (VERIFICATION, sid, v). If $0 = v$, ignore the next steps.
3. Send (POST, $sid, \mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j}) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_I, \mathsf{pk}_{C_j})$) to $\mathcal{F}_{BB}$.

Credential showing:

1. (Credential showing) On input (SHOW-CRED, $sid, \mathsf{pk}_{C_i}, \mathcal{P}_j$), $\mathcal{P}_i$ proves ownership of pk_{C_i} by sending (ZK-PROVER, $sid, \mathcal{P}_j, x, \mathsf{sk}_{C_i}$) to $\mathcal{F}_{ZK}$ where x is a statement for the relation $\text{ZK}_{cred}\{\mathsf{sk}_{C_i} \mid \mathsf{Sig.VerifyKey}(\mathsf{sk}_{C_i}, \mathsf{pk}_{C_i})\}$.
2. (Verify Credential Showing) Any party, upon receiving (ZK-PROOF, sid, x) for ownership of pk_{C_j} from $\mathcal{F}_{ZK}$ additionally verifies that $\mathcal{F}_{BB}$ contains an entry $(\mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j}))$ s.t. $\mathsf{Sig.Verify}(\mathsf{pk}_I, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j})) = 1$, and contains no entry $(\mathtt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$ s.t. $\mathsf{Sig.Verify}(\mathsf{pk}_R, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j})) = 1$.

Privacy Revocation:

1. (Request Privacy Revocation) On input (ANNOUNCE-REV, sid, pk_{C_j}), $\mathcal{I}$ requests privacy revocation for pk_{C_j}, by generating $\sigma_R(\mathsf{pk}_{C_j}) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_R, \mathsf{pk}_{C_j})$ and sending (POST, $sid, (\mathtt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$), to $\mathcal{F}_{BB}$.
2. (Privacy Revocation Response) A user, $\mathcal{P}_i$, observing an entry $(\mathtt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$ on $\mathcal{F}_{BB}$ (with a valid signature):
   (a) If no entries $(\mathtt{req}, \vec{E}_j, \cdot, \vec{q}_j, \mathsf{pk}_{C_j}, \cdot)$ and $(\mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j}))$ exists on $\mathcal{F}_{BB}$, ignore the next steps.
   (b) If for no k, $\mathsf{pk}_{\mathcal{P}_i} = \mathsf{Enc.Decrypt}(\vec{q}_j[k], \mathsf{sk}_{\mathcal{P}_i})$, ignore the next steps.
   (c) Calculates $s_k = \mathsf{Enc.Decrypt}(\vec{E}_j[k], \mathsf{sk}_{\mathcal{P}_i})$.
   (d) Constructs a statement x_k for the relation $\text{ZK}_{rev_k}\{\mathsf{sk}_{\mathcal{P}_i} \mid s_k = \mathsf{Enc.Decrypt}(\vec{E}_j[k], \mathsf{sk}_{\mathcal{P}_i}) \wedge \mathsf{pk}_{\mathcal{P}_i} = \mathsf{Enc.Decrypt}(\vec{q}_j[k], \mathsf{sk}_{\mathcal{P}_i}) \wedge \mathsf{Sign.VerifyKey}(\mathsf{sk}_{\mathcal{P}_i}, \mathsf{pk}_{\mathcal{P}_i})\}$ and sends the message (PROVE, $sid, x_k, \mathsf{sk}_{\mathcal{P}_i}$) to $\mathcal{F}_{NIZK}$ and await response (PROOF, sid, π_{rev_k}).
   (e) Encrypt π_{rev_k} and s_k for the issuer as $\tilde{s}_k \leftarrow \mathsf{Enc.Encrypt}(s_k, \mathsf{pk}_R)$, $\tilde{\pi}_{rev_k} \leftarrow \mathsf{Enc.Encrypt}(\pi_{rev_k}, \mathsf{pk}_R)$.
   (f) Sends a message (REV-SHARE, $sid, \mathsf{pk}_{C_j}, \tilde{s}_k, x_k \tilde{\pi}_{rev_k}$), to $\mathcal{I}$.
3. (Reconstruct Revoked Identity) Upon receiving a message (REV-SHARE, $sid, \mathsf{pk}_{C_j}, \tilde{s}_k, x_k \tilde{\pi}_{rev_k}$), $\mathcal{I}$ uses sk_I to decrypt $\tilde{s}_k$ and $\tilde{\pi}_{rev_k}$, to obtain s_k and π_{rev_k}, and verifies π_{rev_k} by sending (VERIFY, sid, x_k, π_{rev_k}) to $\mathcal{F}_{NIZK}$. On reply (VERIFICATION, $sid, 1$), $\mathcal{I}$ adds s_k to its internal set $S_{\mathsf{pk}_{C_j}-shares}$. If now $|S_{\mathsf{pk}_{C_j}-shares}| = \lceil \frac{n}{2} \rceil + 1$, calculate the revoked identity key by Lagrange interpolating the polynomial f' defined by the shares in $S_{\mathsf{pk}C_j-shares}$ and then calculate the identity public key of the revoked user as $f'(0) = \mathsf{pk}_{\mathcal{P}_j}$.

Fig. 5: $\prod_{PC}$ - Credential Issuance, Credential Showing and Privacy Revocation.

the **Request Privacy Revocation** procedure. This procedure consists of publishing an announcement of the request for privacy revocation, signed with the privacy revocation key. Any (honest) user $\mathcal{P}_i$, observing such a request executes the **Privacy Revocation Response** procedure, where it first checks that a credential exists for this credential in step (a). If so, in step (b) all committee member indicators in $\vec{q_j}$ of that request are checked by decrypting them with the responding users identity secret key $\mathsf{sk}_{\mathcal{P}_i}$. If decryption results in the users identity public key $\mathsf{pk}_{\mathcal{P}_i}$ for the k'th indicator, $\mathcal{P}_i$ holds the k'th seat in the committee. If so, it (c) decrypts the k'th share, (d) proves correct decryption and committee membership, and (e) encrypts both the share and proof (since the proof reveals the share) for the issuer, and (f) sends the ciphertexts to the issuer. The issuer, when receiving such a share, executes the **Reconstruct Revoked Identity** procedure to decrypt and check the proof. When it has obtained a majority of the shares, it reconstructs the revoked identity and obtains $\mathsf{pk}_{\mathcal{P}_j}$.

4.1 Security Analysis of $\prod_{PC}$

We now prove that $\prod_{PC}$ realizes $\mathcal{F}_{PC}$ in the presence of a static malicious adversary capable of corrupting up to $\frac{m}{2} - 1$ users.

Theorem 1. *Let* Sig *be a signature scheme,* BSig *be a blind signature scheme,* SPSig *be a structure preserving signature scheme,* SShare *be a* $(t, n)-$*threshold secret sharing scheme,* C *be a commitment scheme,* Enc *be a key-private* $\mathsf{IND\text{-}CPA}$*-secure public-key encryption scheme and* Shuf *be a zero-knowledge proof of shuffle correctness. Protocol* $\prod_{PC}$ *UC-realizes* $\mathcal{F}_{PC}$ *in the* $(\mathcal{F}_{BB}, \mathcal{F}_{PKI}, \mathcal{F}_{ZK}, \mathcal{F}_{NIZK})$*-hybrid model with security against a static active adversary* $\mathcal{A}$ *corrupting a minority of* $\mathcal{P}_1, \ldots, \mathcal{P}_m$ *such that a committee of size* $n \leq m$ *has honest majority with overwhelming probability.*

Proof. Let $\mathcal{A}$ be a static adversary allowed to corrupt up to $m/2-1$ parties before the start of the execution, which remain corrupt throughout the execution. We prove Theorem 1 by showing that for each $\mathcal{A}$, there exists a simulator $\mathcal{S}_{PC}$ so that any environment $\mathcal{Z}$ has a negligible advantage in determining whether it is interacting with $\mathcal{A}$ and $\prod_{PC}$ or $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$. $\mathcal{S}_{PC}$ is described in Figures 6 and 7.

Indistinguishably of Setup The vectors $\vec{f}$ ($\vec{f}[j] = \mathsf{C.Commit}(\vec{\mathsf{pk}}_{\mathcal{P}}[j], 1)$) and $\vec{f_i'}$ ($(\vec{f_i'}, \pi_{\rho_i}) = \mathsf{Shuf.Prove}(m, \rho_i, \vec{f})$) are indistinguishable from those computed in a real execution due to the hiding property of commitments. Similarly, π_{ρ_i} is indistinguishable due to the zero knowledge property of zero knowledge proofs. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably of Credential Requests The simulated proof π_{esc_i} is indistiguishable from the one computed in a real execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{NIZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably Credential Issuance Here the creation of a credential is simulated without having any information about the identity of the honest party who requests the credential in the real world execution.

Indistinguishably of Credential Showings $\mathcal{S}_{PC}$ simulates the showing of a credential without having any information about the identity of the honest party who shows it in the real world execution. (ZK-PROOF, sid, x) is indistinguishable from the one computed in the real world execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{ZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably of Privacy Revocation When simulating honest users responses to privacy revocation requests, π_{rev_k}, computed for the adjusted shares s'_k, is indistinguishable from the one computed in the real world execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{NIZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Notice that throughout the simulation $\mathcal{S}_{PC}$ interacts with $\mathcal{A}$ exactly as an honest party would in $\prod_{PC}$, except when simulating credential issuance and showing for honest parties. In these cases, $\mathcal{S}_{PC}$ simulates the creation of a credential and its showing without having any information about the identity of the honest party who requests/shows the credential. However, this is indistinguishable from the real world execution since these proofs are done via $\mathcal{F}_{NIZK}$ and $\mathcal{F}_{ZK}$, which produces messages distributed exactly as in a real world execution. Moreover, by extracting witnesses from proofs done by $\mathcal{A}$ via $\mathcal{F}_{NIZK}$ and $\mathcal{F}_{ZK}$, $\mathcal{S}_{PC}$ activates $\mathcal{F}_{PC}$ with inputs that match $\mathcal{A}$'s behavior. Hence, $\mathcal{Z}$ cannot distinguish the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$. □

5 From Static to Proactive Security

Protocol $\prod_{PC}$ as described in the previous sections realizes a PAPR credential scheme using efficient building blocks, in the static security setting. In this section, we sketch how to construct proactively secure PAPR Credentials, at the price of using less efficient building blocks.

Maintaining the revocation committee secret in the presence of a mobile adversary naturally puts us in the YOSO setting: the identities of committee members must remain anonymous, so before they act in a revocation process (or before) the adversary moves, they must re-share the revocation information they hold towards a new anonymous committee. While it would be straightforward to design a protocol realizing $\mathcal{F}_{PC}$ by use of YOSO MPC, it would be terribly inefficient, since it would require computing our credential issuance procedure as part of a very complex YOSO MPC computation where a fresh anonymous committee performs each round. Instead, we propose two alternative and more

Simulator $\mathcal{S}_{PC}$ (First Part)

$\mathcal{S}_{PC}$ interacts with a copy of the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. $\mathcal{S}_{PC}$ forwards all messages between $\mathcal{A}$ and $\mathcal{Z}$. $\mathcal{S}_{PC}$ acts as $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$ towards $\mathcal{A}$, by following their respective descriptions (unless explicitly stated).

Setup:

1. (Issuer Setup) Run the $\prod_{PC}$ procedure for **Issuer Setup** if the issuer is honest and send (SETUP, *sid*) to $\mathcal{F}_{PC}$.
2. (User Setup) Run the $\prod_{PC}$ procedure for **User Setup** for all honest users, then send (SETUP, *sid*) to $\mathcal{F}_{PC}$ for all honest users.
   For each (POST, *sid*, $\cdot$) sent to $\mathcal{F}_{PKI}$ by a corrupt party, send (SETUP, *sid*) to $\mathcal{F}_{PC}$.
3. (Hide Committee Candidates) Run the **Hide Committee Candidates** procedure for each honest user.
4. (Sample Committee) If the issuer is honest, then for each (POST, $sid, (\texttt{hide}, \vec{f}'_j, \pi_{\rho_j}, \mathsf{pk}_{T_j}, \sigma_{T_j}(\vec{f}'_j), \sigma_B(\mathsf{pk}_{T_j}))$ sent to $\mathcal{F}_{BB}$, (either by a corrupt $\mathcal{P}_j$ or when simulating an honest $\mathcal{P}_j$), the simulator executes step (1) of the **Sample Committee** procedure. If the checks verify, also execute step (2).

Credential Request: If the simulator receives a message (CRED-REQ, *sid*) from $\mathcal{F}_{PC}$, an honest user has requested a credential. To simulate this, $\mathcal{S}_{PC}$ executes the **Credential Request** procedure in $\prod_{PC}$, but does so using arbitrary values for $\mathsf{sk}_{\mathcal{P}_i}, \mathsf{pk}_{\mathcal{P}_i}$ and $\sigma_E(\mathsf{pk}_{\mathcal{P}_i})$ and a simulated proof π_{esc_i} for the arbitrary values. If later $\mathcal{S}_{PC}$ receives a message for $\mathcal{F}_{NIZK}$, (VERIFY, sid, x_i, π_{esc_i}) it responds with (VERIFICATION, *sid*, 1) instead of following the $\mathcal{F}_{NIZK}$ description. If the simulator receives $(\texttt{req}, \vec{E}_i, \vec{f}'_i, \vec{q}_i, \mathsf{pk}_{C_i}, x_i, \pi_{esc_i})$ from a corrupt user $\mathcal{P}_i$, intended for $\mathcal{F}_{BB}$, the simulator executes checks in step (1) and (2) of the **Credential Issuance** procedure. If the checks clear, the request is valid and the simulator then sends (CRED-REQ, *sid*) to $\mathcal{F}_{PC}$, awaits the message (KEY?, *sid*) and responds by sending (KEY, sid, pk_{C_i}) to $\mathcal{F}_{PC}$.

Credential Issuance: If $\mathcal{I}$ is honest, on message (CRED, sid, pk_{C_j}) from $\mathcal{F}_{PC}$ simulate by executing step (3) of the **Credential Issuance** procedure. If instead $\mathcal{I}$ is corrupt, for each message (POST, $sid, \texttt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j})$) sent to $\mathcal{F}_{BB}$, $\mathcal{S}_{PC}$ sends (ISSUE-CRED, sid, pk_{C_j}) to $\mathcal{F}_{PC}$ if $\sigma_I(\mathsf{pk}_{C_j})$ is a valid signature on pk_{C_j} by $\mathcal{I}$.

Credential showing:

1. (Credential showing) Whenever the simulator receives (VALID-CRED, $sid, \mathsf{pk}_{C_i}, \mathcal{P}_j$) from $\mathcal{F}_{PC}$, $\mathcal{S}_{PC}$ simulates a successful show of credential pk_{C_i} to party $\mathcal{P}_j$ by simulating a successful proof of relation ZK_{cred} via $\mathcal{F}_{ZK}$ with $\mathcal{P}_j$ acting as verifier, resulting in $\mathcal{P}_j$ receiving (ZK-PROOF, sid, x_i) from $\mathcal{F}_{ZK}$.
2. (Verify Credential Showing) Whenever the simulator receives a message (ZK-PROVER, $sid, \mathcal{P}_j, x_i, \mathsf{sk}_{C_i}$), where x_i is a statement for $\text{ZK}_{cred}\{\mathsf{sk}_{C_i} \mid \mathsf{Sig.VerifyKey}(\mathsf{sk}_{C_i}, \mathsf{pk}_{C_i})\}$, from a corrupt party $\mathcal{P}_i$, intended for $\mathcal{F}_{ZK}$, it verifies the proof by following the description of $\mathcal{F}_{ZK}$. If the verification clears, send (SHOW-CRED, $sid, \mathsf{pk}_{C_i}, \mathcal{P}_j$) to $\mathcal{F}_{PC}$.

Fig. 6: Simulator $\mathcal{S}_{PC}$ for protocol $\prod_{PC}$.

efficient constructions. The first demonstrates how to wrap our protocol $\prod_{PC}$

Simulator $\mathcal{S}_{PC}$ (Second Part)

Privacy Revocation:

1. (Request Privacy Revocation) If the issuer is honest, and the simulator receives a message $(\text{ANNOUNCE-REV}, sid, \mathsf{pk}_{C_j})$ from $\mathcal{F}_{PC}$, it executes the **Request Privacy Revocation** procedure in $\prod_{PC}$ to simulate the issuer. If the issuer is corrupt and $\mathcal{S}_{PC}$ receives $(\text{POST}, sid, \texttt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$, intended for $\mathcal{F}_{BB}$ where $\sigma_R(\mathsf{pk}_{C_j})$ is a valid revocation signature on pk_{C_j}, the simulator sends $(\text{ANNOUNCE-REV}, sid, \mathsf{pk}_{C_j})$ to $\mathcal{F}_{PC}$.
2. (Reconstruct Revoked Identity) When $\mathcal{S}_{PC}$ receives a message with the identity of a user $(\text{IDENTITY}, sid, \mathcal{P}_i, \mathsf{pk}_{C_i})$, it must simulate responses from the honest committee members. If $\mathcal{I}$ is honest or $\mathcal{P}_i$ is corrupt, $\mathcal{S}_{PC}$ thus executes the $\prod_{PC}$ procedure for **Privacy Revocation Response** for each honest $\mathcal{P}_j$.
   If $\mathcal{I}$ is corrupt and $\mathcal{P}_i$ is honest the simulator needs to "adjust" the shares which the honest committee members respond with, so that the shares reconstruct to $\mathsf{pk}_{\mathcal{P}_i}$ rather that the arbitrary value used during simulation of the credential request. Therefore the simulator first constructs a polynomial f of degree $\lceil \frac{n}{2} \rceil - 1$ where $f(\mathsf{pk}_{\mathcal{P}_i}) = 0$ and $f(k) = \vec{s}_i[k]$ for each k where $\rho_i(k) \in \vec{b}_i$ (*i.e.* for the corrupt users in the simulated committee, don't change the shares).
   Then for each k where $\rho_i(k) \notin \vec{b}_i$ (honest users) let $s'_k = f(k)$ and construct a statement x_k for relation ZK_{rev_k} with s'_k. If later the simulator receives a message for $\mathcal{F}_{NIZK}$, $(\text{VERIFY}, sid, x_k, \pi_{rev_k})$ it responds with $(\text{VERIFICATION}, sid, 1)$ instead of following the $\mathcal{F}_{NIZK}$ description. Honest committee members are then simulated by executing step (e) and (f) of the **Privacy Revocation Response** procedure using s'_k and π_{rev_k} as constructed by the simulator.

Finally, when $\mathcal{A}$ stops, output whatever $\mathcal{A}$ outputs to $\mathcal{Z}$.

Fig. 7: Simulator $\mathcal{S}_{PC}$ for protocol $\prod_{PC}$.

with a YOSO resharing procedure to obtain proactive security. The second improves efficiency further by using YOSO Threshold Encryption directly.

5.1 Modeling Proactive Security

We model proactive security, similarly to [47], by each party in the system having an *epoch* tape which maintains an integer `epoch` initialized to 0 at the start of the execution. The execution proceeds in phases which alternate between an *operational* phase and a *refreshing* phase, starting with the operational phase. In contrast to [47], we force every party to have the same value as epoch counter.

Epochs The refreshing stage is started by the adversary sending `refresh` to *all* parties. Refresh of individual parties is not allowed. Upon receiving the `refresh` command, a party increases `epoch` by 1 and executes its instructions for refreshment. Once each party has completed its refreshment instructions and handed over execution to $\mathcal{Z}$, a new operational phase begins.

Corruptions A mobile adversary $\mathcal{A}$ can corrupt or uncorrupt any party $\mathcal{P}_i$ *after* a refreshing phase ends (*i.e.* after the last party has handed over execution to $\mathcal{Z}$) but *before* the next operational phase starts (*i.e.* before the first activation of a party in the operational phase). After $\mathcal{A}$ moves, every party $\mathcal{P}_i$ remains corrupted (or honest) throughout that entire operational phase. At no time can $\mathcal{A}$ corrupt more than $\lceil \frac{m}{2} \rceil - 1$ parties.

5.2 Proactive Security Through YOSO Resharing

Let us now describe how to modify $\prod_{PC}$ to obtain proactive security by adding a re-sharing procedure in the YOSO model. Resharing is a standard procedure in proactive secret sharing that allows a set of parties to transfer a shared secret for which they hold shares to a second set of parties who obtain fresh shares independent from the original ones. On a high level, YOSO resharing allows for a current committee to reshare a secret towards a future anonymous committee while only speaking once. Such a YOSO resharing procedure can be added to our PAPR protocol without modifying existing procedures. That is, we use $\prod_{PC}$ as it is, but add a YOSO reshare procedure for maintaining the escrowed user identities over different epochs. Before every new epoch starts, current revocation committees reshare the identity information they hold towards a single anonymous committee that holds this information in the next epoch. We refer to this protocol as $\prod_{PC-P}$. The approach is illustrated in Figure 8.

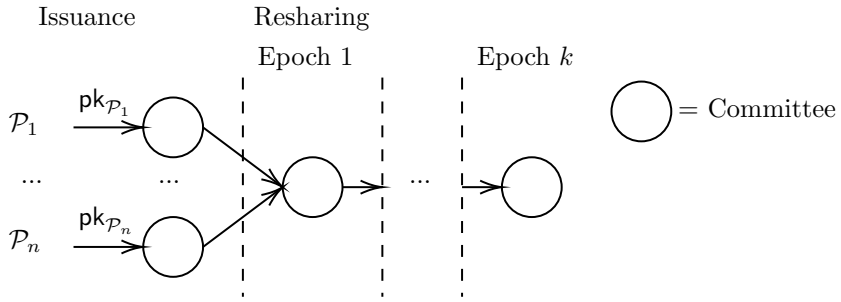

Fig. 8: Functioning of $\prod_{PC-P}$ with YOSO resharing: as in the issuance procedure of $\prod_{PC}$, initially each user $\mathcal{P}_i$ secret shares its identity $\mathsf{pk}_{\mathcal{P}_i}$ towards a different designated hidden committee. Subsequently, before the start of each epoch, the committees reshare the identities towards a new single anonymous committee.

A YOSO resharing scheme can be abstractly described as having a **committee establishment** part, where all parties jointly elect the new committee without learning it, and a **resharing** part, where the current committee provably reshares the committee secret to the new committee without learning or revealing the new committee members. Multiple choices are available for implementing YOSO resharing, *e.g.* Evolving-Committee Proactive Secret Sharing

[11], Random-Index Private Information Retrieval [39] plus standard resharing techniques, or YOLO YOSO Anonymous Committee PVSS Resharing [26]. We refrain from picking a particular scheme, and instead use the **committee establishment** and **resharing** procedures abstractly, as described below:

Committee Establishment During committee establishment, a single committee for the next epoch of size n is elected from all m committee candidates, without revealing the committee to *any party*. This procedure will output a set of anonymous public keys which constitute the committee keys.

Resharing During resharing, each member of the current epoch committee re-shares the secret using the anonymous public keys of the next epoch's committee. This procedure will thus output a set of anonymously encrypted shares of the secret. Before these encrypted shares are published, the old shares must be made inaccessible, *e.g.* by deleting them.

Figure 9 describes how to add a refresh procedure based on YOSO-Resharing to $\prod_{PC}$ in order to realize $\mathcal{F}_{PC}$ proactive security against a mobile adversary $\mathcal{A}$. Protocol $\prod_{PC-P}$ is obtained by executing $\prod_{PC}$ with the modifications described in Figure 9 in order to securely refresh shares of revocation information across epoch changes. We here indicate instances of functionalities specific to an epoch be indicated in the superscript, so that $\mathcal{F}_{PKI}^{1}$ is the shared instance during the first epoch and $\mathcal{F}_{PKI}^{2}$ the shared instance during the second.

Hold Revocation Responses: Postpone revocation requests until *refresh* phase.
Reshare: $\mathcal{P}_i$ on command `refresh` from $\mathcal{Z}$ does:

(a) Generate new keys $(\mathsf{pk}'_{\mathcal{P}_i}, \mathsf{sk}'_{\mathcal{P}_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$, replace $\mathcal{F}_{PKI}^{\mathtt{epoch}}$ with $\mathcal{F}_{PKI}^{\mathtt{epoch\ +\ 1}}$ and send $(\text{POST}, sid, \mathsf{pk}'_{\mathcal{P}_i})$ to $\mathcal{F}_{PKI}^{\mathtt{epoch\ +\ 1}}$.
(b) Execute the YOSO **Committee Establishment** procedure, obtaining the anonymous committee public keys for the `epoch + 1` committee.
(c) For each postponed revocation request for credentials issued in the current epoch, execute steps (1) to (5) of **Privacy Revocation Response** in $\prod_{PC}$, *i.e.* stopping before sending shares to $\mathcal{I}$.
(d) If $\mathcal{P}_i$ is part of the YOSO committee for the current epoch, handle any revocation requests for credentials issued during previous epochs by executing steps (3) to (5) of the **Privacy Revocation Response** procedure in $\prod_{PC}$.
(e) Erase $\mathsf{sk}_{\mathcal{P}_i}$. Set $\mathsf{sk}_{\mathcal{P}_i} = \mathsf{sk}'_{\mathcal{P}_i}$, $\mathsf{pk}_{\mathcal{P}_i} = \mathsf{pk}'_{\mathcal{P}_i}$ and $\mathtt{epoch} = \mathtt{epoch} + 1$.
(f) For all credentials that have not been revoked, execute YOSO resharing of escrowed identities towards the `epoch + 1` committee. For all revocation requests handled in steps (c) or (d), post the results by executing step (6) of the **Privacy Revocation Response** procedure in $\prod_{PC}$.

Wrap: Any other input is forwarded to $\prod_{PC}$.

Fig. 9: Sketch of proactive security wrapper protocol $\prod_{PC-P}$.

Assuming an ideal functionality $\mathcal{F}_{YPSS}$ capturing YOSO proactive secret sharing with the properties outlined above, the security of $\prod_{PC-P}$ is captured as follows. Notice that such a $\mathcal{F}_{YPSS}$ can be obtained via the techniques of [38, 39, 26] plus UC-secure NIZKs modelled $\mathcal{F}_{NIZK}$.

Theorem 2. *(Informal) Let* Sig *be a signature scheme,* BSig *be a blind signature scheme,* SPSig *be a structure preserving signature scheme,* SShare *be a* $(t, n)-$*threshold secret sharing scheme,* C *be a commitment scheme,* Enc *be a key-private* $\mathsf{IND\text{-}CPA}$*-secure public-key encryption scheme and* Shuf *be a zero-knowledge proof of shuffle correctness. Protocol* $\prod_{PC-P}$ *UC-realizes* $\mathcal{F}_{PC}$ *in the* ($\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$, $\mathcal{F}_{NIZK}$, $\mathcal{F}_{YPSS}$)*-hybrid model, with proactive security against a mobile active adversary* $\mathcal{A}$ *corrupting a minority of parties in* $\mathcal{P}_1, \ldots, \mathcal{P}_m$ *so that any committee of size* $n \leq m$ *has honest majority, with overwhelming probability.*

5.3 Proactive Security Through YOSO Threshold Encryption

While the protocol in Figure 9 shows how to wrap $\prod_{PC}$ with a YOSO-resharing step to obtain proactive security, it is possible to realize a proactively secure PAPR credential scheme in a more efficient way using YOSO Threshold Encryption [35]. We can realize a PAPR Credential scheme assuming we have such a YOSO Threshold encryption system, with procedures for setting up YOSO committees (*Committee Selection*), generating a committee keypair so that all system parties hold the public key and each committee member holds a share of the corresponding secret key (*Distributed Key Generation*), resharing the secret key (*Reshare*), decryption of a ciphertext to a share of the plaintext (*Threshold Decryption*) and reconstruction of the plaintext given a sufficient amount of shares of the plaintext (*Reconstruct*). We sketch our protocol $\prod_{PC-PT}$ below:

Setup Each party $\mathcal{P}_i$ generates an identity keypair and registers the public key on a PKI. The issuer $\mathcal{I}$ generates issuance and revocation keypairs, registers the public keys on a PKI and publishes signatures of each user's public key under the issuance key. All $\mathcal{P}_i$ execute the *Committee Selection* and the anonymous committee executes the *Distributed Key Generation* procedure obtaining a threshold public key pk_{THE} and shares of the corresponding secret key.

Credential Issuance To request a credential, a user generates a new credential keypair, encrypts its identity public key under pk_{THE}. It then sends this ciphertext and the public key of the new credential keypair to the issuer over an anonymous channel and proves in zero knowledge that it knows the private key and issuer signature on the encrypted public key. If the issuer accepts the proof, it returns a signature on the credential public key.

Revocation Request The issuer requests privacy revocation for a credential by signing the credential public key with its revocation key and posting the signature on a bulletin board.

Reshare and Revocation Response On command `refresh` from $\mathcal{Z}$, all current epoch honest committee members constructs revocation responses for privacy revocation requests correctly posted on the system bulletin board by executing the *Threshold Decryption* procedure to obtain shares of the revoked users

identity public key. They then execute the committee *Reshare* procedure before giving the shares to the issuer. When the issuer obtain these shares, it learns the identity key of the revoked user by executing the *Reconstruct* procedure.

Assuming an ideal functionality $\mathcal{F}_{YTHE}$ capturing YOSO threshold encryption with the properties outlined above, the security of $\prod_{PC-PT}$ is captured as follows. Notice that such a $\mathcal{F}_{YTHE}$ can be obtained via the techniques o [35] by employing UC-secure NIZKs as modelled in $\mathcal{F}_{NIZK}$ and UC-secure proactive resharing as modelled in $\mathcal{F}_{YPSS}$ (discussed above).

Theorem 3. *(Informal) Let* Sig *be a signature scheme,* BSig *be a blind signature scheme and* Enc *be a key-private* IND-CPA*-secure public-key encryption scheme. Protocol* $\prod_{PC-PT}$ *UC-realizes* $\mathcal{F}_{PC}$ *in the (*$\mathcal{F}_{BB}$*,* $\mathcal{F}_{PKI}$*,* $\mathcal{F}_{ZK}$*,* $\mathcal{F}_{NIZK}$*,* $\mathcal{F}_{YTHE}$*)-hybrid model with proactive security against a mobile active adversary* $\mathcal{A}$ *corrupting a minority of* $\mathcal{P}_1, \ldots, \mathcal{P}_m$ *such that a committee of size* $n \leq m$ *has honest majority with overwhelming probability.*

The advantage of this approach in relation to the simple extension $\prod_{PC-P}$ using YOSO resharing is that using YOSO threshold encryption in this way gives us amortized communication complexity essentially independently from the number of credentials issued. Notice that in $\prod_{PC-P}$ the YOSO committees are required to hold shares of the identity public keys connected to every credential that has been issued (and not revoked). On the other hand, in this improved construction, the YOSO committees only need to hold shares of the secret key for the threshold encryption scheme. Moreover, credential issuance also becomes cheaper, since a party who requests a credential no longer needs to secret share its identity public key towards a committee. In the new credential issuance procedure, a party only needs to publish an encryption of its identity public key under the threshold encryption public key, which also makes the zero-knowledge proof it generates in this phase cheaper (*i.e.* proving that a single ciphertext contains a certain message, instead of proving that a set of encrypted secret shares reconstruct that message).

6 Practical Considerations

We now discuss the properties of PAPR for anonymous credential schemes from a practical perspective.

6.1 Optimizing the Committee Size

Given a set of parties $\mathcal{P}$ of size m and a certain corruption ratio t, we are interested in sampling uniformly at random the minimum number of parties n from $\mathcal{P}$ such that an honest majority committee is guaranteed with overwhelming probability $1-2^{-\kappa}$, where κ is a security parameter. This situation is extensively described in [33], but to aid intuition we here provide a few numerical examples when $\kappa = 60$. If $m = 10,000$ and $t = 30\%$, then $n = 462$. If $m = 2,000$ and $t = 30\%$, then $n = 382$. If $m = 10,000$ and $t = 20\%$, then $n = 178$. If $m = 2,000$ and $t = 20\%$, then $n = 164$.

6.2 Flexibility in the Protocol Design

Throughout the paper we made some simplifying assumptions to ease the explanation. Below, we discuss ways to generalize our protocol in the cases where the assumptions are not actual limitations of the protocol design.

Multiple Authorities The $\mathcal{F}_{PC}$ functionality and its concrete realization, $\prod_{PC}$, are defined for a single issuer $\mathcal{I}$. This is done to keep the protocol simple and easy to read. Extending the scheme to multiple authorities can be done straightforwardly in two ways. One way is to exploit the fact that the scheme is proven to be universally composable, so we can run multiple parallel instances without compromising security. This approach requires no changes to the functionality or the protocol description. A second way is to define $\mathcal{F}_{PC}$ for multiple issuing parties. This can be done by imposing that credential requests shall specify which $\mathcal{I}$ that can issue and revoke the credential, and by letting credential showings be valid for any issuing $\mathcal{I}$. This change can be trivially reflected in our $\prod_{PC}$ construction.

Separating the Issuance and Revocation Roles Analogously to the previous paragraph, we have kept the protocol description simple by appointing a single party $\mathcal{I}$ for both issuance and revocation roles. Modifying $\mathcal{F}_{PC}$ and $\prod_{PC}$ by introducing a revoking party $\mathcal{R}$, and appointing the privacy revocation role to $\mathcal{R}$, rather than $\mathcal{I}$, is straightforward: In $\mathcal{F}_{PC}$ allow $\mathcal{R}$ (instead of $\mathcal{I}$) to send $(\text{ANNOUNCE-REV}, sid, \cdot)$. In $\prod_{PC}$ move the generation and PKI-registration of the revocation keypair $(\mathsf{pk}_R, \mathsf{sk}_R)$ into a separate **Revoker Setup** procedure, and in the **Privacy Revocation Response** procedure, send the shares to $\mathcal{R}$ rather than to $\mathcal{I}$. This separation of roles can be combined with the above modification for multiple authorities to freely select a desired set of issuers and revokers.

Establishing Eligible Committee Candidates In PAPR, the set of committee candidates is the root of trust for the guaranteed privacy revocation and public announcement. In practice, our system can easily be adapted to have the list of eligible committee candidates be publicly chosen and endorsed, *e.g.*, through an election or by the issuer. In particular, the set of committee candidates does not have to coincide with the whole set of users.

Separating Users and Committee Candidates $\prod_{PC}$ is described assuming the set of users and the set of committee candidates to be the same. Indeed, $\prod_{PC}$ can be modified to accommodate a set of committee candidates that is independent from the set of users. For instance, split $\mathcal{P}$ into a subset $\mathcal{C} = \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_c}\}$ of potential committee members and a subset of standard users $\mathcal{U} = \mathcal{P} \setminus \mathcal{C}$, and run the instructions **Hide Committee Candidates** and **Sample Committee** (from Figure 3), letting the index run among the public keys in $\mathcal{C}$. In such a separation, committee candidates may be expected to be online all the time. This

behavior can be incentivized through a reward system or by law constraints. On the other hand, users are allowed to be offline whenever they wish.

Managing a dynamic user set $\prod_{PC}$ crucially relies on $\mathcal{F}_{PKI}$ to contain a fixed list of all parties before credentials are issued. In practice the set of active users might however change over time, with users joining or leaving the system. However, this reliance is not as strong as it appears on first glance.

By running parallel instances of $\prod_{PC}$ with multiple authorities, as described above, each new instance will have a separate $\mathcal{F}_{PKI}$. Thus users joining an already existing system can be enrolled to a new instance of the protocol.

On the other hand, if enough committee candidates leave the system, e.g. due to loss of their keys, the possibility of privacy revocation can be affected. While a party leaving the system would technically fall under corrupt behaviour, this is not a problem in $\prod_{PC-P}$ and $\prod_{PC-PT}$. This is since these protocols reshare committee secrets and explicitly use a new instance of $\mathcal{F}_{PKI}$ for each epoch. Thus, inactive users will not enroll with the new $\mathcal{F}_{PKI}$ and will as a consequence not be considered committee candidates anymore. In the case of $\prod_{PC}$ however, this mechanism is not present, and one must therefore account for the probability of parties leaving the system when selecting the size of n.

6.3 Overhead From a User Perspective

Despite the many parts of the protocol, from a user perspective, the protocol is a very low cost endeavor. $\prod_{PC}$ is designed with user overhead in mind, reducing complexity for the user and keeping as much of the resulting complexity in the credential issuance phase. A user only needs to store a bare minimum of their own identity key and their own credentials. Credential issuance is somewhat computationally intense for the user, but this only happens once – per credential issuance. During normal (application) operation, there is *zero* computational overhead for the user. Finally, a user will experience some additional computational overhead *when and only if* they are involved as a committee member in an actual privacy revocation request (or in a YOSO-resharing for $\prod_{PC-P}$). So in summary, computational efforts for users are only necessary in the beginning and sometimes (or rarely) at the end of an epoch, but never during normal operation.

6.4 Practical Attacks

Denial of Service An adversary with the capability to mount large scale Denial of Service (DoS) attacks, *i.e.* targeting all potential committee members, can of course delay privacy revocation while the attack is maintained. However, it cannot prevent revocation indefinitely. Once the DoS attack is mitigated or no longer maintained, the protocol can simply resume execution, at which point the identity of the user will be revealed. Since the committee members are revealed to the user during credential issuance, one can also imagine DoS attacks targeting only the committee members by a corrupt user utilizing this knowledge. However, while such an attack is cheaper to mount, it is not feasible to maintain it indefinitely. Thus, DoS attacks can delay, but not prevent privacy revocations.

Sybil Attacks Sybil attacks, where a single party poses a multiple parties, are prevented due to the fact that each user needs to enroll (*i.e.* post to $\mathcal{F}_{PKI}$) in the system with a public key linked to their real identity. Thus we obtain a list of the actual users in the system, preventing Sybil attacks.

6.5 Towards an Efficient Instantiation of PAPR Credentials

We here provide a list of building blocks that may be used to efficiently instantiate our $\prod_{PC}$ protocol.

- To prove correct shuffling of committee candidates' public keys, the Bayer and Groth's scheme [8] may be used, and the computational complexity for the prover is $O(m \log(\sqrt{m}))$, where m is the number of committee candidates.
- For Sig, Boneh Boyen signatures may be used [13, Section 4.3], where the computational complexity is constant for both signing and verifying.
- For SPSig, Abe et al.'s scheme $\mathsf{SIG1}$ in [3, Section 4.1] may be used, where the complexity is linear in the size of the message, which in our case makes it constant since in our protocol we only sign single group elements.
- For Enc and C, ElGamal encryption may be used, in the second case we see ciphertexts as commitments and rely on the schemes' binding property.
- Protocols realizing the functionalities $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$ can be found in [25, 51, 23, 43], respectively.

As described at a high level in Figure 4, ZK_{esc}, which is at the core of our protocol, proves the following.

- ① ZK_{ID} states that the user is the owner of $\mathsf{pk}_{\mathcal{P}}$, *i.e* it knows the secret key $\mathsf{sk}_{\mathcal{P}}$, *and* knows a signature generated by the issuer on $\mathsf{pk}_{\mathcal{P}}$, *i.e* $\sigma_E(\mathsf{pk}_{\mathcal{P}})$. Thus the computational complexity to prove it is constant $\mathcal{O}(1)$.
- ② ZK_{share} states that (2.1) the n shares are constructed correctly, *i.e.* any set of k shares will reconstruct to the users public key $\mathsf{pk}_{\mathcal{P}}$. Further, (2.2) each of these shares, (2.3) is correctly encrypted, (2.4) for the correct committee member, (2.5) which is correctly indicated in $\vec{q}$. Each of these steps introduces a computational complexity that is linear with respect to n.

The overall complexity of ZK_{esc} is therefore $\mathcal{O}(n)$.

We additionally provide a discussion of heuristically substituting functionalities for non-UC but more efficient building blocks in Appendix C.

References

1. Abadi, M., e.a.: An open letter from us researchers in cryptography and information security (Jan 2014), `http://masssurveillance.info/`
2. Abdalla, M., Namprempre, C., Neven, G.: On the (im)possibility of blind message authentication codes. In: CT-RSA 2006 (Feb 2006)
3. Abe, M., Chase, M., David, B., Kohlweiss, M., Nishimaki, R., Ohkubo, M.: Constant-size structure-preserving signatures: Generic constructions and simple assumptions. Journal of Cryptology (4) (Oct 2016)

4. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. Journal of Cryptology (2) (Apr 2016)
5. Badertscher, C., Gazi, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: ACM CCS 2018 (Oct 2018)
6. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: CRYPTO 2017, Part I (Aug 2017)
7. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: ACM CCS 2013 (Nov 2013)
8. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: EUROCRYPT 2012 (Apr 2012)
9. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: P-signatures and noninteractive anonymous credentials. In: TCC 2008 (Mar 2008)
10. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: ASIACRYPT 2001 (Dec 2001)
11. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: TCC 2020, Part I (Nov 2020)
12. Blömer, J., Bobolz, J.: Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In: ACNS 18 (Jul 2018)
13. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. Journal of Cryptology (2) (Apr 2008)
14. Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In: CRYPTO'93 (Aug 1994)
15. Camenisch, J., Drijvers, M., Dubovitskaya, M.: Practical UC-secure delegatable credentials with attributes and their application to blockchain. In: ACM CCS 2017 (Oct / Nov 2017)
16. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable and modular anonymous credentials: Definitions and practical constructions. In: ASIACRYPT 2015, Part II (Nov / Dec 2015)
17. Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: Efficient periodic n-times anonymous authentication. In: ACM CCS 2006 (Oct / Nov 2006)
18. Camenisch, J., Lehmann, A.: (Un)linkable pseudonyms for governmental databases. In: ACM CCS 2015 (Oct 2015)
19. Camenisch, J., Lehmann, A.: Privacy-preserving user-auditable pseudonym systems. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 269–284. IEEE (2017)
20. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: EUROCRYPT 2001 (May 2001)
21. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: CRYPTO 2004 (Aug 2004)
22. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS (Oct 2001)
23. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Report 2002/140 (2002), `https://eprint.iacr.org/2002/140`
24. Cascudo, I., David, B.: SCRAPE: Scalable randomness attested by public entities. In: ACNS 17 (Jul 2017)

25. Cascudo, I., David, B.: ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In: ASIACRYPT 2020, Part III (Dec 2020)
26. Cascudo, I., David, B., Garms, L., Konring, A.: YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. Cryptology ePrint Archive, Report 2022/242 (2022), https://eprint.iacr.org/2022/242
27. Chaum, D.: Blind signatures for untraceable payments. In: CRYPTO'82 (1982)
28. Chaum, D., van Heyst, E.: Group signatures. In: EUROCRYPT'91 (Apr 1991)
29. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theoretical Computer Science **777**, 155–183 (2019)
30. Crites, E.C., Lysyanskaya, A.: Delegatable anonymous credentials from mercurial signatures. In: CT-RSA 2019 (Mar 2019)
31. Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: FC 2019 (Feb 2019)
32. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: EUROCRYPT 2018, Part II (Apr / May 2018)
33. David, B., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: GearBox: An efficient UC sharded ledger leveraging the safety-liveness dichotomy. Cryptology ePrint Archive, Report 2021/211 (2021), https://eprint.iacr.org/2021/211
34. Daza, V., Haque, A., Scafuro, A., Zacharakis, A., Zapico, A.: Mutual accountability layer: accountable anonymity within accountable trust. In: International Symposium on Cyber Security, Cryptology, and Machine Learning. pp. 318–336. Springer (2022)
35. Erwig, A., Faust, S., Riahi, S.: Large-scale non-interactive threshold cryptosystems through anonymity. Cryptology ePrint Archive, Report 2021/1290 (2021), https://eprint.iacr.org/2021/1290
36. Frankle, J., Park, S., Shaar, D., Goldwasser, S., Weitzner, D.J.: Practical accountability of secret processes. In: USENIX Security 2018 (Aug 2018)
37. Garay, J.A., MacKenzie, P.D., Yang, K.: Strengthening zero-knowledge protocols using signatures. Journal of Cryptology (2) (Apr 2006)
38. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You only speak once - secure MPC with stateless ephemeral roles. In: CRYPTO 2021, Part II (Aug 2021)
39. Gentry, C., Halevi, S., Magri, B., Nielsen, J.B., Yakoubov, S.: Random-index PIR and applications. In: TCC 2021, Part III (Nov 2021)
40. Goldwasser, S., Park, S.: Public accountability vs. secret laws: Can they coexist? a cryptographic proposal. In: Proceedings of the 2017 on Workshop on Privacy in the Electronic Society. pp. 99–110 (2017)
41. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. In: Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I (2022)
42. Green, M., Kaptchuk, G., Laer, G.V.: Abuse resistant law enforcement access systems. In: EUROCRYPT 2021, Part III (Oct 2021)
43. Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. Journal of the ACM (JACM) **59**(3), 1–35 (2012)
44. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: EUROCRYPT 2008 (Apr 2008)
45. Hellman, M.: Open letter to Senator Ron Wyden (Feb 2018)
46. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO 2017, Part I (Aug 2017)

47. Kondi, Y., Magri, B., Orlandi, C., Shlomovits, O.: Refresh when you wake up: Proactive threshold wallets with offline devices. In: 2021 IEEE Symposium on Security and Privacy (May 2021)
48. Laurie, B., Langley, A., Kasper, E., Messeri, E., Stradling, R.: Certificate Transparency Version 2.0. RFC 9162 (Dec 2021). https://doi.org/10.17487/RFC9162, https://www.rfc-editor.org/info/rfc9162
49. Lindell, Y., Lysyanskaya, A., Rabin, T.: On the composition of authenticated byzantine agreement. In: 34th ACM STOC (May 2002)
50. Lueks, W., Everts, M.H., Hoepman, J.H.: Vote to link: Recovering From Misbehaving Anonymous Users. In: Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society. pp. 111–122 (2016)
51. Masny, D., Watson, G.J.: A PKI-based framework for establishing efficient MPC channels. In: ACM CCS 2021 (Nov 2021)
52. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: Bringing key transparency to end users. In: USENIX Security 2015 (Aug 2015)
53. Micali, S.: Fair Cryptosystems. Tech. rep., Massachusetts Institute of Technology (1993)
54. Okamoto, T.: Efficient blind and partially blind signatures without random oracles. In: TCC 2006 (Mar 2006)
55. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery (11) (Nov 1979)
56. Stadler, M.: Cryptographic protocols for revocable privacy. Ph.D. thesis, Verlag nicht ermittelbar (1996)
57. Stadler, M., Piveteau, J.M., Camenisch, J.: Fair blind signatures. In: EUROCRYPT'95 (May 1995)
58. Teranishi, I., Furukawa, J., Sako, K.: k-Times anonymous authentication (extended abstract). In: ASIACRYPT 2004 (Dec 2004)
59. Urs, Gasser, e.a.: Don't panic: Making progress on the "going dark " debate (Feb 2016), https://cyber.harvard.edu/pubrelease/dont-panic/Dont_Panic_Making_Progress_on_Going_Dark_Debate.pdf
60. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: 38th ACM PODC (Jul / Aug 2019)

A Definitions of Standard Primitives

In this section, we present definitions for the syntax of building blocks we use in our protocols and reference their required security guarantees.

Definition 2 (Public Key Encryption Scheme). *A public-key encryption scheme is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *defined as follows.*

$\mathsf{Enc.Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp*.*

$\mathsf{Enc.KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$*: The key generation algorithm takes as input the public parameters and outputs a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

$\mathsf{Enc.Encrypt}(m, \mathsf{pk}) \to c$*: The encryption algorithm takes as input a message* m *and a public key* pk*. It returns a ciphertext* c*.*

Enc.Decrypt$(c, \mathsf{sk}) \to m$*: The decryption algorithm takes as input a ciphertext* c *and a secret key* sk*. It returns a plaintext message* m*.*

We consider an encryption scheme secure if it is indistinguishable under Chosen Plaintext Attacks (IND-CPA) and key-private as formalised in [10, Def. 1].

Definition 3 (Signature Scheme). *A signature scheme Sig with message space* $\mathcal{M}$ *is a tuple of PPT algorithms* (*Setup*, *KeyGen*, *Sign*, *Verify*) *defined as follows.*

Sig.Setup(1^λ)*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp*.*

Sig.KeyGen$(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$*: The key generation algorithm takes as input the public parameters and outputs a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

Sig.Sign$(\mathsf{sk}, m) \to \sigma$*: The sign algorithm takes in input a secret key* sk *and a message* $m \in \mathcal{M}$*; it outputs a signature* σ*.*

Sig.Verify$(\mathsf{pk}, m, \sigma) \to v$*: The verification algorithm takes in input a public key* pk*, a message* $m \in \mathcal{M}$ *and signature* σ*. It outputs 0 (reject) or 1 (accept).*

Throughout the paper we assume Sig to be *correct* and *existentially unforgeable* as in [13, Def. 2]. In a nutshell this means that signatures generated by the signing algorithm are always accepted by the verifying algorithm, and that without the knowledge of the secret key it is computationally infeasible to generate a signature that is accepted by the verifying algorithm with non-negligible probability.

Definition 4 (Blind signature). *A blind signature BSig with message space* $\mathcal{M}$ *is a tuple of PPT algorithms* (*KeyGen*, *User*, *Sign*, *Verify*) *defined as follows ([2]).*

BSig.KeyGen$(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$*: The randomized key generation algorithm takes as input a security parameter* 1^λ *with* $\lambda \in \mathbb{N}$ *and outputs a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

BSig.User and BSig.Sign are randomized interactive algorithms.

The user runs BSig.User on an initial state (pk, m)*, where* pk *is a public key and* $m \in \mathcal{M}$ *is a message, and let it interact with BSig.Sign run by the signer on initial state a secret key* (sk)*. At the end of the protocol, BSig.User either enters the* `halt` *state and outputs a signature* σ *as its last outgoing message, or enters the* `fail` *state. Instead, BSig.Sign simply enters the* `halt` *state, without generating any output.*

BSig.Verify$(\mathsf{pk}, m, \sigma) \to v$*: The deterministic verification algorithm takes in input a public key* pk*, a message* $m \in \mathcal{M}$ *and signature* σ*. It outputs 0 (reject) or 1 (accept).*

Where an *interactive algorithm* is a stateful algorithm that on input an incoming message m_{in} (this is ϵ if the party is initiating the protocol) and state St outputs an outgoing message m_{out} and updated state St'. Two interactive algorithms `A` and `B` are said to *interact* when the outgoing messages of `A` are passed as incoming messages to `B`, and vice versa, until both algorithms enter either the `halt` or `fail` state. We write $(m_{\mathtt{A}}, St_{\mathtt{A}}, m_{\mathtt{B}}, St_{\mathtt{B}}) \leftarrow [\mathtt{A}(St_{\mathtt{A}}) \leftrightarrow \mathtt{B}(St_{\mathtt{A}})]$

to denote the final state after an interaction between $\mathtt{A}$ and $\mathtt{B}$ when run on initial states $St_{\mathtt{A}}$ and $St_{\mathtt{B}}$, respectively.

(Correcteness) A blind signature scheme is *correct* if for all $\lambda \in \mathbb{N}$ and for all $m \in \mathcal{M}$, it holds that $St_{\mathsf{BSig.User}} = \mathtt{halt}$ and $\mathsf{BSig.Verify}(\mathsf{pk}, m, \sigma) \to 1$ when $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ and $(m_{\mathsf{Sign}}, St_{\mathsf{Sign}}, \sigma, St_{\mathsf{User}}) \leftarrow [\mathsf{BSig.Sign}(\mathsf{sk}) \leftrightarrow \mathsf{BSig.User}((\mathsf{pk}, m))]$ with probability 1.

A blind signature has to guarantee *unforgeability*, *i.e.*, a user should not be able to forge signatures, and *blindness*, *i.e.*, the signer should not be able to see the messages that is being signed, or even be able to relate signed messages to previous protocol sessions.

(Unforgeability) Let $\mathsf{BSig} = (\mathsf{KeyGen}, \mathsf{User}, \mathsf{Sign}, \mathsf{Verify})$ be a blind signature scheme, let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathcal{A}$ be a forging algorithm with access to the signing oracle. The experiment $\mathbf{Exp}^{\mathrm{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$ first generates a keypair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ and runs $\mathcal{A}$ on input $(1^\lambda, \mathsf{pk})$. The adversary has access to a signing oracle that runs the $\mathsf{BSig.Sign}(\mathsf{sk}, .)$ algorithm and maintains state across invocations. At the end of its execution, the adversary outputs a set of message signatures pairs $\{(m_1, \sigma_1), \ldots, (m_s, \sigma_s)\}$. Let t be the number of completed signing sessions during $\mathcal{A}$'s attack. Then $\mathcal{A}$ is said to win the game if $\mathsf{BSig.Verify}(\mathsf{pk}, m_i, \sigma_i) \to 1$ for all $1 \leq i \leq s$, all m_i are different and $s > t$. BSig is said to be *unforgeable* (*one-more unforgeable under sequential attacks*) if the probability of winning the above game is negligible for all PPT adversaries $\mathcal{A}$.

Formally, the experiment is defined as follows:

Experiment $\mathbf{Exp}^{\mathrm{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$:
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$
$SSet \leftarrow \emptyset$; $s \leftarrow 0$ // set of signer sessions and number of finished sessions
$\{(m_1, \sigma_1), \ldots, (m_s, \sigma_s)\} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk} : \mathsf{Sign}(.,.))$
if $\mathsf{BSig.Verify}(\mathsf{pk}, m_i, \sigma_i) \to 1$ for all $1 \leq i \leq s$ and $s \geq$ t and $m_i \neq m_j$ for all $1 \leq i < j \leq s$ **then**
  Return 1
else
  Return 0
end if

Where $\mathcal{A}$'s queries to the signing oracles are answered as follows:

Oracle $\mathsf{Sign}(s, m_{\mathrm{in}})$: // s is a session identifier
if $s \notin SSet$ **then**
  $SSet \leftarrow \{s\}$; $St_{\mathsf{BSig.Sign}}[s] \leftarrow \mathsf{sk}$
  $(m_{\mathrm{out}}, St_{\mathsf{BSig.Sign}}[s]) \leftarrow \mathsf{BSig.Sign}(m_{\mathrm{in}}, St_{\mathsf{BSig.Sign}}[s])$
  if $St_{\mathsf{BSig.Sign}}[s] = \mathtt{halt}$ **then**
    $t \leftarrow t + 1$
  end if

Return m_{out}
end if

The advantage of $\mathcal{A}$ in breaking BSig is defined as the probability that the above experiment returns 1:

$$\mathbf{Adv}^{\text{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = Pr[\mathbf{Exp}^{\text{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = 1]$$

BSig is said to be *one-more unforgeable under sequential attacks* if the advantage $\mathbf{Adv}^{\text{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$ is a negligible function in the security parameter λ for all PPT adversaries $\mathcal{A}$.

(Blindness) Let $\mathsf{BSig} = (\mathsf{KeyGen}, \mathsf{User}, \mathsf{Sign}, \mathsf{Verify})$ be a blind signature scheme, let $\lambda \in \mathbb{N}$ be the security parameter and let $\mathcal{A}$ an adversary. The adversary $\mathcal{A}$ acts as a cheating signer, who is trying to distinguish between two signatures created in different signing sessions. The experiment chooses a random bit b, generates a fresh key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ and runs the adversary $\mathcal{A}$ on input $(1^\lambda, \mathsf{pk}, \mathsf{sk})$. $\mathcal{A}$ outputs two challenge messages m_0 and m_1 and then act as the signer in two sequential interactions with the $\mathsf{BSig.User}$ algorithm. If $b = 0$, then $\mathcal{A}$ first interacts with $\mathsf{BSig.User}(\mathsf{pk}, m_0)$ and then with $\mathsf{BSig.User}(\mathsf{pk}, m_1)$; If $b = 1$, then $\mathcal{A}$ first interacts with $\mathsf{BSig.User}(\mathsf{pk}, m_1)$ and then with $\mathsf{BSig.User}(\mathsf{pk}, m_0)$. If in both sessions the $\mathsf{BSig.User}$ algorithms accept, then $\mathcal{A}$ is additionally given the resulting signatures σ_0, σ_1 for messages m_0, m_1. Finally, $\mathcal{A}$ outputs its guess d and wins the game if $b = d$. BSig is said to be *blind* (*blind under sequential attacks*) if $2p - 1$, where p is the probability that $\mathcal{A}$ wins the above game, is negligible for all PPT adversaries $\mathcal{A}$.

Formally, the experiment is defined as follows:

Experiment $\mathbf{Exp}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$:
$b \leftarrow \{0,1\}$; $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$
$((m_0, m_1), St_{\mathcal{A}}) \leftarrow \mathcal{A}(\epsilon, (1^\lambda, \mathsf{pk}, \mathsf{sk}))$
$(m_{\mathcal{A}}, St_{\mathcal{A}}, \sigma_b, St_b) \leftarrow [\mathcal{A}(St_{\mathcal{A}}) \leftrightarrow \mathsf{User}((\mathsf{pk}, m_b))]$
$(m_{\mathcal{A}}, St_{\mathcal{A}}, \sigma_{1-b}, St_{1-b}) \leftarrow [\mathcal{A}(St_{\mathcal{A}}) \leftrightarrow \mathsf{User}((\mathsf{pk}, m_{1-b}))]$
if $St_0 = \texttt{fail}$ or $St_1 = \texttt{fail}$ **then**
  $\sigma \leftarrow \texttt{fail}$
else
  $\sigma \leftarrow (\sigma_0, \sigma_1)$
end if
$d \leftarrow \mathcal{A}(\sigma, St_{\mathcal{A}})$
if b = d **then**
  Return 1
else
  Return 0
end if

The advantage of $\mathcal{A}$ in breaking BSig is defined as

$$\mathbf{Adv}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = 2 \cdot \Pr[\mathbf{Exp}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = 1] - 1$$

BSig is said to be *blind under sequential attacks* if the advantage $\mathbf{Adv}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$ is a negligible function in the security parameter λ for all PPT adversaries $\mathcal{A}$.

Definition 5 (Commitment Scheme). *A commitment scheme C is a tuple of PPT algorithms (Setup, Commit, Open) defined as follows.*

C.Setup(1^λ)*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp *(implicit input in all subsequent algorithms)*

C.Commit$(m, r) \to c$*: This procedure takes as input a vector of messages* $m = \{m_1, \ldots, m_n\}$ *and a vector of randomness* $r = \{r_1, \ldots, r_n\}$ *(sometimes referred to as key). It outputs a commitment* c*.*

C.Open$(c, m, r) \to v$*: This procedure takes as input a commitment* c*, a vector of messages* $m = \{m_1, \ldots, m_n\}$ *and a vector of randomness* $r = \{r_1, \ldots, r_n\}$*. It outputs a verification bit indicating whether* m, r *is an opening to* c *or not.*

A commitment scheme should be *correct*, i.e the opening procedure will return 1 (accept) with probability 1 if the commitment c is generated by Commit on the remainder of the input to Open. Furthermore, it should be *hiding*, in the sense that the commitment leaks no information about the message, and *binding* so that the commitment can only be opened to the committed message. These properties are defined in [4, Def. 1, Def. 2].

Definition 6 (Provable Shuffle of Commitments). *A proof system Shuf =* (*Setup*, *Prove*, *Verify*) *for proving shuffle of commitments generated by a commitment scheme C consists of the following algorithms.*

Shuf.Setup(1^λ)*: The setup algorithm takes as input the security parameter and outputs public parameters* pp*, often referred to as the common reference string (implicitly input to all subsequent algorithms).*

Shuf.Prove$(n, \rho, \{c_i\}_{i\in[n]}) \to (\{c'_i\}_{i\in[n]}, \pi)$*: The provable shuffle algorithm takes as input an integer* n*, a permutation* ρ *over the set* $\{1, \ldots, n\}$*, and* n *commitments* $\{c_i\}_{i\in[n]}$ *generated by C.Commit. It returns a list of* n *commitments* $\{c'_i\}_{i\in[n]}$ *and a proof* π*.*

Shuf.Verify$(n, \{c_i\}_{i\in[n]}, \{c'_i\}_{i\in[n]}, \pi) \to v$*: The verification algorithm takes as input an integer* n*, two sets of* n *commitments and a proof* π*. It returns 1 (accept) if* π *is a valid proof for the relation "there exists a set* $M = \{m_i\}_{i\in[n]}$ *and a permutation* $\rho \in S_n$ *s.t.* $\{C.Open(c_i, m_i, r_i)\}_{i\in[n]} = \{C.Open(c'_{\rho(i)}, m_{\rho(i)}, r'_{\rho_i}\}_{i\in[n]}$*", where the randomnesses* r_i, r'_i *are extracted from* π*. Otherwise it returns 0 (reject).*

We assume this scheme is executed by a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, both of which are probabilistic polynomial time interactive algorithms. The public transcript generated by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs s and t is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$ and we write $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = b$, $b \in \{0, 1\}$ for rejection or acceptance. Let R be a polynomial time decidable ternary relation. We denote w as a witness for the statement x if $(\sigma, x, w) \in R$ and we define the languages

$$L_\sigma = \{x \mid \exists w : (\sigma, x, w) \in R\}$$

as the set of statements x having a witness w for the relation R.

The triple $(\mathsf{Shuf.Setup}, \mathcal{P}, \mathcal{V})$ is called an *argument* for a relation R with perfect completeness if for all non-uniform polynomial time interactive adversaries $\mathcal{A}$ we have:

(Computational soundness)

$$\Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : x \leftarrow \mathcal{A}(\sigma, \text{hist}) : x \notin L_\sigma \wedge \langle \mathcal{A}, \mathcal{V}(\sigma, x)\rangle = 1] \approx 0$$

(Perfect completeness)

$$\Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : x \leftarrow \mathcal{A}(\sigma, \text{hist}) : (\sigma, x, w) \notin R \vee \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x)\rangle = 1] = 1$$

Moreover, an argument $(\mathsf{Shuf.Setup}, \mathcal{P}, \mathcal{V})$ is called *public coin* if the verifier chooses their messages uniformly at random and independently of the messages sent by the prover, *i.e.*, the challenges correspond to the verifier's randomness ρ. Then we define:

(Perfect special honest verifier zero knowledge) A public coin argument $(\mathsf{Shuf.Setup}, \mathcal{P}, \mathcal{V})$ is called *perfect special honest verifier zero knowledge* (SHVZK) argument for R with common reference string generator $\mathsf{Shuf.Setup}$ if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} &\Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : (x, w\rho);\\ tr \leftarrow \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x, \rho)\rangle &: (\sigma, x, w) \in R \wedge \mathcal{A}(tr) = 1]\\ &= \Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : (x, w\rho);\\ &\quad tr \leftarrow \mathcal{S}(\sigma, x, \rho) : (\sigma, x, w) \in R \wedge \mathcal{A}(tr) = 1] \end{aligned}$$

Then, to construct a fully zero-knowledge argument secure against arbitrary verifiers one can first construct a perfect special honest verifier zero knowledge argument and then convert it into a fully zero-knowledge argument [37].

A scheme with the above properties can be efficiently realized from the proof of shuffle correctness for ciphertexts of [8]. In our setting, we view an ElGamal ciphertext as a commitment (since it is unconditionally binding and computationally hiding) and use proofs of commitment shuffle correctness to convince a verifier that two distinct sets of commitments yield the same set of openings.

B Functionalities

The bulletin functionality $\mathcal{F}_{BB}$ is defined in Figure 10. It is modified from [25, Fig. 17] to not be authenticated (since we assume anonymous posting to the bulletin board). In the main body of the paper we often omit specifying the MID for brevity.

$\mathcal{F}_{BB}$ keeps an initially empty ordered list $\mathcal{M}$ and interacts with a set of parties $\mathcal{P}$ as follows:

Post to Bulletin Board: Upon receiving a message $(\text{POST}, sid, MID, m)$ from a party $\mathcal{P}_i \in \mathcal{P}$, if there is no message $(sid, MID, m') \in \mathcal{M}$, append (sid, MID, m) to the list $\mathcal{M}$ of messages that were posted in the public bulletin board. Then send $(\text{POSTED}, sid, MID, m)$ to $\mathcal{S}$.

Read from Bulletin Board: Upon receiving a message (READ, sid) from a party in $\mathcal{P}$, return $(\text{READ}, sid, \mathcal{M})$ to the caller.

Fig. 10: Ideal functionality $\mathcal{F}_{BB}$.

Next, we present the $\mathcal{F}_{PKI}$ functionality, defined in Figure 11, which is taken from [51, Figure 3] with a slight change of wording.

$\mathcal{F}_{PKI}$ keeps an initially empty list $\mathcal{M}$ of messages and interacts with a set of parties $\mathcal{P}$ as follows:

Report Query: Upon receiving a message (REPORT, sid, v) from party $\mathcal{P}_i$, record $(\mathcal{P}_i, v)$ in $\mathcal{M}$ iff $\mathcal{P}_i$ does not have a record and no record containing v exists.

Retrieve Query: Upon receiving a message $(\text{RETRIEVE}, sid, \mathcal{P}_i)$, look up and reply with $(\mathcal{P}_i, v)$, where $v = \perp$ when there is no record for $\mathcal{P}_i$.

Fig. 11: Ideal functionality $\mathcal{F}_{PKI}$.

The $\mathcal{F}_{ZK}$ functionality in Figure 12 is adapted from [23, Fig. 7], so that the identity of the prover is not revealed to the verifier. In practice, this can be realized by communicating over a sender-anonymous channel.

Finally, we give the non-interactive zero knowledge functionality $\mathcal{F}_{NIZK}$ in Figure 13, taken from [43, Fig. 4].

$\mathcal{F}_{ZK}$ proceeds as follows, running with the parties in $\mathcal{P}$, and parameterized with a relation R:

Prove: Upon receiving $(\text{ZK-PROVER}, sid, \mathcal{P}_j, x, w)$ from $\mathcal{P}_i$: If $R(x, w) = 1$, then send the message $(\text{ZK-PROOF}, sid, x)$ to $\mathcal{P}_j$ and $\mathcal{S}$. Otherwise, ignore.

Fig. 12: Ideal functionality $\mathcal{F}_{ZK}$.

Parameterized with relation R and running with the parties in $\mathcal{P}$.

Proof: On input $(\text{PROVE}, sid, x, w)$ from party $\mathcal{P}_i$ ignore if $(x, w) \notin R$. Send (PROVE, sid, x) to $\mathcal{S}$ and wait for answer (PROOF, sid, π). Upon receiving the answer store (x, π) and send (PROOF, sid, π) to $\mathcal{P}_i$.

Verification: On input $(\text{VERIFY}, sid, x, \pi)$ from $\mathcal{P}_j$ check whether (x, π) is stored. If not send $(\text{VERIFY}, sid, x, \pi)$ to $\mathcal{S}$ and wait for an answer $(\text{WITNESS}, sid, w)$. Upon receiving the answer, check whether $(x, w) \in R$ and in that case, store (x, π). If (x, π) has been stored return $(\text{VERIFICATION}, sid, 1)$ to $\mathcal{P}_j$, else return $(\text{VERIFICATION}, sid, 0)$.

Fig. 13: Ideal functionality $\mathcal{F}_{NIZK}$.

C Heuristics for Efficient Subtitutions of Functionalities

To instantiate $\prod_{PC}$ efficiently without Universal Composability, the ideal functionalities $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$ may be substituted respectively by a blockchain such as Ethereum (note that $\mathcal{F}_{BB}$ may also be implemented starting from consensus protocols such as those in [29, 46, 32, 31, 5, 6, 49, 60]), a PKI with key transparency such as CONIKS [52], Schnorr proofs over the Tor network and Groth-Sahai proofs [44]. We stress that the security of these substitutions would be heuristic. If formally proven secure, the resulting scheme would at best be proven *sequentially* composable, due to the nature of Groth-Sahai proofs.

In such a system where $\mathcal{F}_{NIZK}$ is substituted for Groth-Sahai proofs, the conditions (2.3) and (2.4) in ZK_{esc} (Figure 4) can be realized as the verification equations of a pairing-based PVSS scheme, e.g. [24].