Foundations of Anonymous Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions

Jan Bobolz[1], Jesus Diaz[2], and Markulf Kohlweiss[3]

[1]University of Edinburgh, UK
[2]Input Output, Spain
[3]University of Edinburgh and Input Output, UK

January 10, 2024

Abstract

In today's systems, privacy is often at odds with utility: users that reveal little information about themselves get restricted functionality, and service providers mistrust them. In practice, systems tip to either full anonymity (e.g. Monero), or full utility (e.g. Bitcoin). Well-known cryptographic primitives for bridging this gap exist: anonymous credentials (AC) let users disclose a subset of their credentials' attributes, revealing to service providers "just what they need"; group signatures (GS) allow users to authenticate anonymously, to be de-anonymized "just when deemed necessary". However, these primitives are hard to deploy.

Current AC and GS variants reach specific points in the privacy-utility tradeoff, which we point as counter-productive engineering-wise, as it requires full and error-prone re-engineering to adjust the tradeoff. Also, so far, GS and AC have been studied separately by theoretical research.

We take the first steps toward unifying and generalizing both domains, with the goal of bringing their benefits to practice, in a flexible way. We give a common model capturing their core properties, and use functional placeholders to subsume intermediate instantiations of the privacy-utility tradeoff under the same model. To prove its flexibility, we show how concrete variants of GS, AC (and others, like ring signatures) can be seen as special cases of our scheme – to which we refer as *universal anonymous signatures* (UAS). In practice, this means that instantiations following our construction can be configured to behave as variant X of a GS scheme, or as variant Y of an AC scheme, by tweaking a few functions.

Contents

1 Introduction

Anonymous signatures aim to strike a balance between the utility of authenticating identity information and the privacy offered by unlinkability. Exploring different privacy-vs-utility tradeoffs has been at the core of well-known cryptographic primitives for decades. Anonymous credentials (AC) [Cha85] allow users to selectively disclose attributes to verifiers. Group signatures (GS) [CvH91] let users prove group membership, preserving anonymity unless an authority de-anonymizes them. Related schemes, like ring signatures or direct anonymous attestation, share their goal: preserving privacy while authenticating some useful information.

Utility at authentication time. AC schemes focus on offering utility at the moment in which a user shows possession of a credential, often enabling selective attribute disclosure, or arbitrary predicates – e.g. "I am over 18 and from an EU country." In contrast, GS schemes create signatures over arbitrary messages that, in addition, prove the statement "I am a valid member of this group."

Utility after authentication time. Camenisch et al. [CL01, CDL+13] mention AC schemes with conditional release of information after authentication – but, to the best of our knowledge, no formal model is provided, and this type of utility is not frequent in AC. In contrast, GS schemes emphasize utility after signing time, typically allowing trusted parties to open signatures for signer identification.

Utility at issuance time. Frequently, the term utility refers to information revealed by signatures or the authentication process, but some AC schemes add extra semantics to issuance. For instance, some schemes support using previously obtained credentials to request new ones [CDL+13]. In the GS literature, as far as we know, extended behavior at issuance time has not been considered so far.

Why is this not ideal? Many AC and GS variants with different privacy-vs-utility tradeoffs exist, covering many use cases. Yet, real world adoption is limited, with exceptions like Hyperledger Anoncreds [WG23]. We explore potential reasons:

From a security point of view, while reference models exist such as the *foundations of group signatures* series [BMW03, BSZ05, BCC+16], each variant requires a slightly different model to capture privacy and unforgeability properties that deviate from the established tradeoff. That is: whenever we aim for a new privacy-vs-utility tradeoff, a slightly different security model needs to be created, which is not a trivial task.

Engineering-wise, GS or AC schemes usually seem a good fit for privacy problems. But often, the privacy-vs-utility tradeoff needed is not exactly what existing schemes and implementations offer. Then, engineers face a trilemma: (1) if, luckily, a model and provably secure construction – but no implementation – exist for the desired tradeoff, they can implement it from scratch; (2) if an implementation for a closely related scheme exists, they can adapt it *ad hoc*; or (3) they may be forced to abandon the privacy-enhancing approach, to achieve the needed utility. (1) and (2) are error-prone and discouraged for production-ready systems; and (3) is bad for privacy. In another frequent setting, what engineers are demanded for v1 of their product may differ from what their v2 will need. While a flexible system may not always offer the most efficient implementation, it may still be more acceptable than rebuilding the solution from scratch.

Given these concerns, can we create a unified *tradeoff-dynamic* model spanning privacy and security of AC and GS schemes? Is there a generic construction for such a model, offering engineers flexibility to choose their desired privacy-vs-utility tradeoff without requiring full reimplementation, redesign, and proof?

1.1 Our contributions

A model with functional placeholders for dynamic privacy-vs-utility tradeoffs (Section 4). Customization is desirable during credential issuance, authentication, and after authentication. We use functional placeholders to capture possible tradeoffs, bringing the expressiveness of AC show predicates to issuance and opening. Modeling this flexibility requires abstracting anonymity, unforgeability, and non-frameability for issuance and signing. Usually, schemes prove knowledge of a key pair and credential(s), plus a tradeoff-dependent claim that can be captured by a function. Yet, capturing security and privacy with dynamic tradeoffs is harder than with static ones. E.g., in the dynamic case, one can define a function f such that, for two users with keys upk_1, upk_2, and credentials $\mathbf{crd}_1, \mathbf{crd}_2$, $y = f(upk_1, \mathbf{crd}_1) = f(upk_2, \mathbf{crd}_2)$. Thus, user upk_1 producing a signature that opens to y may not be a framing of user upk_2. To address it, we use extraction, and check that the extracted values are consistent with what is expected. We call the resulting scheme *Universal Anonymous Signatures* (UAS).

A generic construction (Section 5). We present a generic construction, Π_{UAS}, that we prove secure under our UAS model. We use BBS+ [ASM06, CDL16, TZ23], a variant of CL signatures [CL02], providing randomizable attribute-based credentials. This signature scheme has been used before (e.g.[GL19]) to build Sign-Randomize-Proof GS schemes [BCN+10]. We also draw inspiration from the Sign-Encrypt-Prove approach to GS schemes [BSZ05]: we have the signer encrypt a function of their credential's attributes under an opener's public key and prove its correct computation. For issuance-time utility, the user proves, during an interactive protocol with the issuer, that a predefined function of their public key and credentials is acceptable.

Relationships with other schemes (Section 6). We study our UAS model and Π_{UAS} construction as a generalization of privacy-preserving signature and authentication schemes. We define several function combinations that instantiate specific privacy-vs-utility tradeoffs within our UAS framework – which we call Π_{UAS} *restrictions*. We prove that these restrictions imply well-known schemes, including digital signatures, GS, AC, and ring signatures, under their respective reference models. While we establish concrete connections for a few schemes, the space of possible Π_{UAS} restrictions is extensive. We give a glimpse of these more advanced connections in Appendix E, where we sketch proofs on how to build GS with message-dependent opening [EHK+19], multimodal private signatures [NGSY22a], and revocable ACs [CKS10]. Further restrictions can be easily imagined, giving schemes such as AC with auditability, ring signatures with some sort of linkability, etc. This allows engineers to adapt privacy-vs-utility tradeoffs by modifying restriction functions, maintaining security and controlling information leakage.

Before presenting our main construction, Section 2 introduces related work on closely related primitives, and Section 3 summarizes our construction's main building blocks. Further details are deferred to the appendix.

Why do we need flexible privacy-vs-utility tradeoffs? A promising use case for digital identity is compliance for global decentralized financial infrastructures (e.g., *Zcash* or *Monero*), or in Centrally Banked Digital Currencies [KKS22]. As the legal frameworks for such systems is still evolving, it is paramount that asset privacy be configurable [Esp22]. UAS offers a principled way to achieve this.

2 Related Work

There are many anonymous signatures (AC, GS, or related) schemes that aim at achieving a different privacy-vs-utility tradeoff. In GS schemes the focus is on opening and linkability: [SEH+12] makes de-anonymization dependent on messages; [GL19] does not allow de-anonymization, but signatures by the same signer are linkable; in [MSS06] signers are fully identified towards other group members and linkable for non group members; in [LNPY21], signers are de-anonymizable only if a predicate of their "identity" and the signed message is not satisfied. In AC schemes, the usual selective disclosure [CL01] is augmented to revealing arbitrary predicates on the credentials' attributes, e.g., in [DMM+18], which is the state of the art in utility at authentication time. Some works consider delegation capabilities [BCC+09, CKLM14], or revocation [CKS10]. We now focus on schemes that aim at more flexible tradeoffs and at general security models that are scheme independent.

Related work on flexible tradeoffs. To the best of our knowledge, our work is the first to model flexible tradeoffs at all steps of the "credential and signature lifecycle". However, some works already pushed towards achieving more flexibility. Benoît et al. and Nguyen et al. [LNPY21, NGSY22a] introduce the notion of what can be called "functional opening". That is, the information that the opener can learn is *a function of* the signer's identity, rather than the identity itself. However, their notion of "identity" is left abstract, which makes it hard to apply in real world settings. We give a concrete definition of identity, via credentials with attributes. Kohlweiss et al. [KLN23] introduces generic functions for utility at opening time, that allow an auditor to learn a function of the user's credential and private information fixed by the auditor in advance.

Related work on general security models. Probably, the most relevant works towards achieving a common and generic model are those in the "Foundations of Group Signatures" line [BMW03, BSZ05, BCC+15], with [KY06] proposed in parallel to [BSZ05]. Before them, many different models coexisted, each focusing on similar but slightly different security and privacy properties. In the AC domain no similar foundational line exists as far as we know, although [CKL+15] does a great job in subsuming previous works and proposing a modular approach towards AC schemes. In some sense, our goal is similar to these unifying works, but focusing on the achieved privacy-vs-utility tradeoff. As mentioned, there are currently many variants of both GS and AC schemes offering similar but slightly different tradeoffs, at the cost of introducing many similar but slightly different models. Our goal is to avoid that.

3 Preliminaries

Public-Key Encryption has Setup, KG, Enc, and Dec algorithms. Setup(1^κ) produces public parameters par. KG(par) generates a encryption-decryption key pair (ek, dk), Enc(ek, m) encrypts

message m with ek and outputs ciphertext c. $\mathrm{Dec}(dk, c)$ decrypts ciphertexts using dk to retrieve message m. We rely on chosen plaintext secure public-key encryption schemes (IND-CPA).

Non-Interactive Zero-Knowledge (NIZK). A NIZK scheme [SCO+01] for a NP relation $\mathcal{R}$ has three algorithms: $\mathrm{Setup}^{\mathcal{R}}$, $\mathrm{Prove}^{\mathcal{R}}$, $\mathrm{Verify}^{\mathcal{R}}$. Algorithm $\mathrm{Setup}^{\mathcal{R}}(1^\kappa)$ produces the common reference string crs. $\mathrm{Prove}^{\mathcal{R}}(crs, x, w)$ creates a NIZK proof π of knowledge of witness w for x such that $(w, x) \in \mathcal{R}$. $1/0 \leftarrow \mathrm{Verify}^{\mathcal{R}}(crs, x, \pi)$ verifies the proof. The properties we build on are completeness, soundness, zero-knowledgeness, and extractability. We require extractability to hold in the presence of a simulator (simulation extractability), and zero-knowledge to hold in the presence of an extractor (extraction zero-knowledge) [GO14].

Signatures over Blocks of Committed Messages, with proofs. We use schemes that allow signing blocks of messages, and commitments to blocks of messages, and which are also compatible with proof systems over the produced signature and signed (commitments to) messages. An SBCM scheme is as a tuple (Setup, KG,Blind, Sign, Unblind,Verify). $par \leftarrow \mathrm{Setup}(1^\kappa)$ produces some public parameters. $(vk, sk) \leftarrow \mathrm{KG}(par)$ produces a verification-signing key pair. $c \leftarrow \mathrm{Blind}(vk, \overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$ is run by a user to request a signature over $\overline{\boldsymbol{msg}} \cup \boldsymbol{msg}$, where $\boldsymbol{msg}$ are revealed to the signer, but $\overline{\boldsymbol{msg}}$ are signed in committed form. $\beta \leftarrow \mathrm{Sign}(sk, c, \pi, \boldsymbol{msg})$ is run by the signer, to produce a partial signature β over a set of committed messages $\overline{\boldsymbol{msg}}$ and set $\boldsymbol{msg}$. $\sigma \leftarrow \mathrm{Unblind}(vk, \beta, c, r, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$ is run by the user to complete the signer's partial signature β. Finally, $1/0 \leftarrow \mathrm{Verify}(vk, \sigma, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$ verifies a signature σ over message vector $\overline{\boldsymbol{msg}} \cup \boldsymbol{msg}$. An SBCM scheme must be unforgeable and blind.

4 Formalizing UAS

In a UAS scheme, users generate their key pair, and optionally advertise their public key and conditions for issuance in order to become issuers. Openers first generate their key pairs, and then advertise what information they expect to learn from signatures. Users may later use their credentials to request new ones, or to produce a signature. Issuance only succeeds if the user's credentials (if any) meet the issuer's requirements. Signatures may directly output some information derived from the user's data. Also, each signature has a selected opener, who can later *learn only the information included by the user*. When openers learn their information, they also prove the correctness of the result, which can be verified by any interested party. This prevents openers from framing innocent users.

4.1 Syntax

In detail, a UAS scheme is composed of the following PPT algorithms:

$\mathrm{Setup}(1^\kappa) \to par$**.** Given security parameter 1^κ, returns global system parameters par. We assume that par are passed implicitly to all other functions.

$\mathrm{KG}(par) \to (upk, usk)$**.** Given par, a user generates a key pair (upk, usk). An *issuer* is a user who defines an issuance function f_{is}. We denote such issuance keys by $ipk = (upk, f_{\mathrm{is}})$ and $isk = (ipk, usk)$.[1]

[1]The (pk, f) tuple simplifies our notation, while simultaneously guaranteeing that users can easily inspect the

OKG(par) $\to$ ($preopk$, $preosk$)**.** An opener runs OKG to generate its pre-opener keys. The opener externally extends the keys with an opening function f_{op}. We denote such opening keys $opk = (preopk, f_{\text{op}})$ and $osk = (opk, preosk)$.

$\langle$Obt($upk, usk, ipk, \boldsymbol{C}, \boldsymbol{a}$), Iss($isk, \boldsymbol{ipk}, \boldsymbol{a}, y_{\text{is}}$)$\rangle \to \langle C/\perp, R/\perp\rangle$**.** Lets a user with key usk obtain a credential $C = (cid, \boldsymbol{a}, crd, ipk)$ from an issuer with key (ipk, isk). cid is a unique identifier for the credential crd, on attribute set $\boldsymbol{a}$. The user may employ previously obtained credentials $\boldsymbol{C} = \{(cid_i, \boldsymbol{a}_i, crd_i, ipk_i)\}_{i\in[n]}$, from which we may omit the ipk_i for readability. The y_{is} value received by the issuer is the claimed output of f_{is}, over the user's data. Note that the issuer can reject initiating the protocol if y_{is} is not acceptable. The user outputs the issued credential C, and the issuer outputs $R \leftarrow (reg, cid)$, where reg is the protocol transcript.

Sign($upk, usk, opk, \boldsymbol{C}, m, f_{\text{ev}}$) $\to (\sigma, y_{\text{ev}})$**.** Upon receiving user secret key usk, opener public key opk, credentials $\boldsymbol{C}$, message m and evaluation function f_{ev}, returns signature σ, and a value y_{ev}. We use Σ to denote the tuple (σ, y_{ev}).

Verify($opk, \boldsymbol{ipk}, \Sigma, m, f_{\text{ev}}$) $\to 1/0$**.** Checks whether $\Sigma = (\sigma, y_{\text{ev}})$ is a valid signature over message m, from a user with credentials issued by issuers with public keys in $\boldsymbol{ipk}$, for evaluation function f_{ev} and opener key opk.

Open($osk, \boldsymbol{ipk}, \Sigma, m, f_{\text{ev}}$) $\to (y_{\text{op}}, \pi)/\perp$**.** Run by the opener with private key osk. Receives a signature $\Sigma = (\sigma, y_{\text{ev}})$ over message m and evaluation function f_{ev}, generated using credentials by issuers with public keys in $\boldsymbol{ipk}$. If Σ is valid, the function outputs a value y_{op}, and a proof of correct opening π.

Judge($opk, \boldsymbol{ipk}, y_{\text{op}}, \pi, \Sigma, m, f_{\text{ev}}$) $\to 1/0$**.** Checks if π is a valid opening correctness proof for the value y_{op}, obtained by applying Open to the the signature $\Sigma = (\sigma, y_{\text{ev}})$ over message m, and for evaluation function f_{ev}.

Issuance, evaluation, and opening functions. These are the functional placeholders modulating the behavior of UAS instantiations. They control the conditions for issuing credentials, the information revealed alongside signatures, and the information revealed when opening signatures.

$f_{\text{is}} : (upk, \boldsymbol{a}, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}) \to y_{\text{is}}$**.** Chosen by each issuer within a family of functions $\mathcal{F}_{\text{is}}$, the issuance function defines the conditions required by the issuer to grant a credential over attributes $\boldsymbol{a}$, when requested by a user with public key upk, optionally using a set of n endorsement credentials with identifiers and attributes given by $\{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}$. The range of f_{is} is R_{is}.

$f_{\text{ev}} : (upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \to y_{\text{ev}}$**.** Signing evaluation functions (or, simply, evaluation functions), from a family of functions $\mathcal{F}_{\text{ev}}$, can be set on a per-signature basis. They receive the user public key upk, a set of credential identifiers and attributes $\{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}$ (where n may be 0), and the message to be signed m. f_{ev} outputs a value y_{ev} from a well defined set R_{ev}.

functions picked by issuers and openers. The second tuple might appear surprising, but it is natural that the secret key in a public key scheme contains at least the information of the public key, e.g., RSA.

$f_{\text{op}} : (upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \to y_{\text{op}}$**.** Chosen by openers from a family of functions $\mathcal{F}_{\text{op}}$. The opening functions define the utility value extractable from signatures. This value is derived from the user's upk, credentials' identifiers and attributes $\{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}$ $(n \geq 0)$ used for signing, and signed message m. It outputs a value y_{op} from a well defined set R_{op}.

For instance, let f_{is} (resp. f_{ev}) output the requesting user's (resp. signer's) public key, and f_{ev} output always 0. The corresponding R_{is} and R_{op} are the set of all possible user public keys, and R_{ev} is $\{0\}$. This combination leads to a Π_{UAS} restriction that behaves like a group signature scheme. See Section 6 for details.

Correctness. An UAS scheme is correct if a signature produced by an honest user, who chooses an honest opener and leverages only credentials obtained from honest issuers, is accepted by an honest verifier, and any opening proof honestly computed from such a valid signature is accepted by an honest judge. Moreover, the y_{ev} (resp. y_{op}) value attached to the signature must match the output of f_{ev} (resp. f_{op}) when evaluated on the endorsement credentials, signed message, and the user's upk. Similarly, the y_{is} value in the transcripts of all involved credentials used to produce the signature must match the output of f_{is} when computed from the requested attributes, user's upk, and any further endorsement credential involved in its issuance. Correctness is formalized in $\mathsf{Exp}^{\text{correct}}_{\mathsf{UAS},\mathcal{A}}$, in Appendix A.

4.2 Security Model

A UAS scheme must satisfy privacy and security properties, both for the issuance protocol, and for the produced signatures. We introduce them semi-formally here. The full formalization as experiments $\mathsf{Exp}^{\text{iss-anon-}b}_{\mathsf{UAS},\mathcal{A}}$, $\mathsf{Exp}^{\text{sig-anon-}b}_{\mathsf{UAS},\mathcal{A}}$, $\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$, $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$, and $\mathsf{Exp}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}$ can be found in Appendix A.

Issuance anonymity. User obtains credentials by running an interactive $\langle \text{Obt}, \text{Iss} \rangle$ protocol with an issuer. The authorization of the issuing can employ previously obtained endorsement credentials to prove that the request meets the issuers requirements—captured by function f_{is} defined by the issuer. We model via the issuance anonymity property that no information about the endorsement credentials besides the output of f_{is} is revealed.

Formally, we define a left-or-right game. The adversary repeatedly plays the issuer in the $\langle \text{Obt}, \text{Iss} \rangle$ protocols against one out of two challenges selected by a random bit b that the adversary needs to guess. Each challenge specifies an honest user and its endorsement credentials. In addition, the adversary can obtain new (non-challenge) credentials for honest users that can be used as endorsement credentials, create signatures with the challenge and non-challenge credentials, open non-challenge signatures, and corrupt users, issuers, and openers at will. To avoid trivial wins, the adversary cannot mix challenge with non-challenge credentials when signing, and the output of the f_{is} function when obtaining challenge credentials and the outputs of f_{ev} and f_{op} functions when signing with challenge credentials need to be the same for both challenge users. To see why this is needed, consider a simple f_{is} function that outputs the public key of the user – which trivially allows an adversarial issuer to distinguish $\langle \text{Obt}, \text{Iss} \rangle$ runs.

Signature anonymity. In UAS, signatures come with signature utility information y_{ev}, computed by f_{ev}, and opener utility information y_{op} computed by f_{op} and only retrievable by the chosen

opener. Signature anonymity captures that signatures do not leak any more information than specified by these functions.

Formally, we define a left-or-right game. The adversary can add honest and corrupt users at will, request signatures using arbitrary credentials, and open the resulting signatures. The adversary is given the capability to repeatedly request signatures for an opener, an evaluation function f_{ev}, a message, and one out of two challenges selected by a random bit b that the adversary needs to guess. Each challenge specifies a challenge users and a set of credentials. To avoid trivial wins, among other simple checks, the y_{ev} value output by f_{ev} has to be the same for both challenges. Similarly, if the opener is corrupt, the value output by f_{op} has to be the same for both challenges. The adversary can open signatures, but only if they are not challenge ones. This is to avoid trivial wins in which the opener was not initially corrupt, and thus the f_{op} equality check did not apply.

Issuance unforgeability. Honest issuers in the interactive $\langle \text{Obt}, \text{Iss} \rangle$ protocol to request new credentials specify an issuing policy f_{is} that involves endorsement credential attributes and identifiers (if any), the requesting user's public key, and the requested attributes. Issuance unforgeability captures that f_{is} must be met, and that the endorsement credentials (if any) have been legitimately obtained.

In a nutshell, the adversary can corrupt users, issuers, and openers, obtain credentials on behalf of honest or corrupt users, and produce signatures leveraging any of these credentials. Eventually, the adversary has to output a credential identifier, which must belong to a credential issued by an honest issuer. The adversary wins if the y_{is} value claimed by the user as an input to Iss does not match the expected output from f_{is}, or if there is a mismatch in the endorsement credentials' attributes or the associated user public key. Note that we require that the credential identifier output by the adversary corresponds to a credential produced by an honest issuer. While we have access to the issuance transcript, we cannot otherwise assume that we know the identifiers of the adversary's endorsement credentials, their attributes, or user public keys needed for evaluating f_{is} and running the required tests. Thus, we resort to extraction: the issuance transcript must allow for the extraction of these otherwise private values.

Note that the issuance unforgeability requirements apply recursively to endorsement credentials. To see this consider adversaries that perform the same oracle queries but output the credential identifiers of endorsement credentials. This excludes construction that allows the use of fraudulently obtained credentials when honestly obtaining a new credential.

Signature unforgeability. Signatures carry two types of utility: y_{ev}, produced by computing f_{ev}, and revealed alongside the signature; and y_{op}, produced by computing f_{op}, and learned by the chosen opener. No adversary should be capable of producing a signature that is accepted by Verify, yet contains *wrong* y_{ev} and y_{op} values. To check this, we let the adversary add corrupt users, issuers, and openers, obtain credentials on behalf of any existing user and issuer, produce signatures by honest users, as well as open any signature. The adversary is challenged to produce a signature, and wins if the signature is accepted by Verify, yet the y_{ev} value associated to it is *wrong*; or if an honest opener cannot produce a proof that is accepted by Judge, or for which the associated y_{op} value is *wrong*. The adversary also wins if any of the credentials used to produce the signature was fraudulently obtained. We define *wrong* by recomputing the f_{ev} and f_{op} functions: as in issuance unforgeability, we extract the necessary inputs from the adversary's signature forgery. We also extract from the issuance transcripts of honestly issued endorsement credentials to check that they were correctly issued.

Figure Fig. 1 explains the need for both issuance and signature unforgeability.

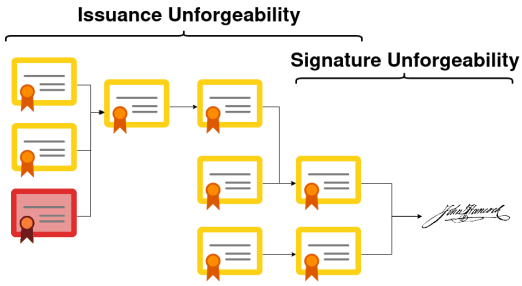

Figure 1: Credential chain segments covered by each unforgeability property. Issuance unforgeability prevents forged credentials at any point, but is agnostic to signatures. Signature unforgeability prevents last-layer credential forgeries, and signature forgeries. E.g., only issuance unforgeability detects if the credential in red is a forgery.

Non-frameability. While the unforgeability properties capture the security expectations of verifiers and openers, non-frameability captures what expectations honest users can still have even when both issuers and openers are corrupt. Concretely, note that in the unforgeability properties, we open using the official opening algorithm. In contrast, in non-frameability the adversary has to produce both a valid signature and valid opening and proof. It wins if this signature was not produced via some of the oracles, if the signature and opening proof are valid, yet the output y_{op} values is *wrong*, or if the *upk* associated to the signature belongs to an honest user. This protects honest users from being framed, either by their *upk* being used to compute y_{op} or by y_{op} being made up in the first place. Again, we make use of extraction techniques to recover all the needed information for attesting correct computation f_{op}, and to learn the signer's *upk*.

5 Π_{UAS}: A Generic UAS Construction

We give a generic construction of a UAS scheme from generic building blocks. We use three NP relations: $\mathcal{R}_{\text{is}}$, $\mathcal{R}_{\text{ev}}$, and $\mathcal{R}_{\text{op}}$, described next. These relations include verifying correct computation of the f_{is}, f_{ev} and f_{op} functions.

$\mathcal{R}_{\text{is}}$: For NIZK proofs at issuance time. Requires users to prove knowledge of their (usk, upk) pair, and the requested credential is bound to *usk*. It also requires any endorsement credential to be a valid credential (signed by some issuer) and bound to *usk*, and enforces the corresponding f_{is} policy.

$\mathcal{R}_{\text{ev}}$: For NIZK proofs at signing time. Ensures that signatures encode the correct signature evaluation (computed via f_{ev}) and opening values (computed via f_{op}), and all the involved credentials are bound to the same *usk*.

$\mathcal{R}_{\text{op}}$: For NIZK proofs at opening time. Ensures that the utility information revealed by the opener, via the Open algorithm, is correct.

$$\mathcal{R}_{\text{is}} = \left\{ \begin{array}{c} (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}), (upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i\in[n]}, r) : \\ (upk, usk) \in [\text{KG}(par)] \wedge \\ c = \text{SBCM.Blind}(ipk, usk, (cid, \boldsymbol{a}), r) \wedge \\ f_{\text{is}}(upk, \boldsymbol{a}, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}) = y_{\text{is}} \wedge \\ \forall i \in [n] : \ \text{SBCM.Verify}(ipk_i, crd_i, usk, (cid_i, \boldsymbol{a}_i)) = 1 \end{array} \right\}$$

$$\mathcal{R}_{\text{ev}} = \left\{ \begin{array}{c} (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), (upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i\in[n]}, y_{\text{op}}, r) : \\ (upk, usk) \in [\text{KG}(par)] \wedge c_{\text{op}} = \text{E.Enc}(ek, y_{\text{op}}; r) \wedge \\ y_{\text{ev}} = f_{\text{ev}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \wedge \\ y_{\text{op}} = f_{\text{op}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \wedge \\ \forall i \in [n] : \ \text{SBCM.Verify}(ipk_i, crd_i, usk, (cid_i, \boldsymbol{a}_i)) = 1 \end{array} \right\}$$

$$\mathcal{R}_{\text{op}} = \left\{ \begin{array}{c} (ek, c, y_{\text{op}}), (dk) : \\ (ek, dk) \in [\text{OKG}(par, \cdot)] \wedge \\ y_{\text{op}} = \text{E.Dec}(dk, c) \end{array} \right\}$$

Figure 2: Specification of the NP relations used in Π_{UAS}. $\mathcal{R} = \{x, w : \text{predicate}(x, w)\}$, where x is the public statement and w is the prover's secret witness.

We build Π_{UAS} as defined in Fig. 3 and Fig. 4. Briefly, Setup computes the public parameters for SBCM, Enc, and the three NIZK systems. KG generates an SBCM user key pair, and OKG an Enc key pair (ek, dk). If a user wishes to *upgrade* itself to an issuer, it sets ($ipk = (upk, f_{\text{is}})$, $isk = (ipk, usk)$) for its chosen f_{is}. Similarly, an opener can upgrade (ek, dk) into ($opk = (ek, f_{\text{op}})$, $osk = (opk, dk)$) by advertising f_{op} in a reliable manner, defining the utility it will accept to extract from signatures. In the $\langle \text{Obt}, \text{Iss} \rangle$ protocol, the user computes f_{is} over the requested attributes, upk, and the endorsement credentials and runs an SBCM blind signature protocol with the issuer, augmented with NIZK.Prove$^{\mathcal{R}_{\text{is}}}$. In Sign, the user computes f_{ev} and f_{op} over the message, attributes, and upk, encrypts y_{op} with the chosen opk, and proves correct computation via NIZK.Prove$^{\mathcal{R}_{\text{ev}}}$. Verify simply verifies the NIZK. In Open, if Verify accepts the signature, the opener decrypts c_{op} to get y_{op}, and outputs it along with a proof obtained via NIZK.Prove$^{\mathcal{R}_{\text{op}}}$. Judge verifies both the signature and the opening proof.

5.1 Correctness and Security of Π_{UAS}

Correctness is by inspection of the honest algorithms. The uniqueness of credentials for honestly issued and obtained credential identifiers is a sub-property of correctness. We observe that both parties contribute uniformly random nonces to this identifier. We give theorems and intuition, but defer formal definitions and proofs to Appendix C.

Theorem 1 (Issuance anonymity of Π_{UAS})**.** *If the* SBCM *scheme is blinding, the NIZK system is zero-knowledge and simulation-extractable, and the public-key encryption scheme is correct and IND-CPA secure, then* Π_{UAS} *satisfies issuance anonymity as defined in Definition 2.*

Theorem 2 (Signature anonymity of Π_{UAS})**.** *If the NIZK system is zero-knowledge and simulation extractable, and the public-key encryption scheme is correct and IND-CPA secure, then* Π_{UAS} *satisfies signature anonymity as defined in Definition 3.*

For the two anonymity properties, we perform game hops until we obtain a game independent of bit b. For this, we simulate the NIZK proofs and replace the encryption of y_{op} with an encryption

$\underline{\text{Setup}(1^\kappa)}$

$par_{\text{SBCM}} \leftarrow \text{SBCM.Setup}(1^\kappa)$
$par_{\text{E}} \leftarrow \text{E.Setup}(1^\kappa)$
$crs_{\text{is}} \leftarrow \text{NIZK.Setup}^{\mathcal{R}_{\text{is}}}(1^\kappa)$
$crs_{\text{ev}} \leftarrow \text{NIZK.Setup}^{\mathcal{R}_{\text{ev}}}(1^\kappa)$
$crs_{\text{op}} \leftarrow \text{NIZK.Setup}^{\mathcal{R}_{\text{op}}}(1^\kappa)$
return $par = (par_{\text{SBCM}}, par_{\text{E}}, crs_{\text{is}}, crs_{\text{ev}}, crs_{\text{op}})$

$\underline{\text{KG}(par)}$

$(par_{\text{SBCM}}, \cdot, \cdot, \cdot, \cdot) \leftarrow par$
$(vk, sk) \leftarrow \text{SBCM.KG}(par_{\text{SBCM}})$
$upk \leftarrow (par, vk)$
$usk \leftarrow sk$
return (upk, usk)

$\underline{\text{OKG}(par)}$

$(\cdot, par_{\text{E}}, \cdot, \cdot, \cdot) \leftarrow par$
$(ek, dk) \leftarrow \text{E.KG}(par_{\text{E}})$
$preopk \leftarrow ek$
$preosk \leftarrow dk$
return $(preopk, preosk)$

$\underline{\text{Sign}(upk, usk, opk, (\{(cid_i, \boldsymbol{a}_i, crd_i, ipk_i)\}_{i\in[n]}), m, f_{\text{ev}})}$

$y_{\text{ev}} \leftarrow f_{\text{ev}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m)$
$(ek, f_{\text{op}}) \leftarrow opk$
$y_{\text{op}} \leftarrow f_{\text{op}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m)$
$c_{\text{op}} \leftarrow \text{E.Enc}(ek, y_{\text{op}}; r)$
$\pi_{\text{ev}} \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), (upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i\in[n]}, y_{\text{op}}, r))$
return $\Sigma = (\sigma = (\pi_{\text{ev}}, c_{\text{op}}), y_{\text{ev}})$

$\underline{\text{Verify}(opk, \boldsymbol{ipk} = \{ipk_i\}_{i\in[n]}, \Sigma, m, f_{\text{ev}})}$

$(\pi_{\text{ev}}, c_{\text{op}}), y_{\text{ev}}) \leftarrow \Sigma$
$(ek, f_{\text{op}}) \leftarrow opk$
return $\text{NIZK.Verify}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, \pi_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek))$

$\underline{\text{Open}(osk, \boldsymbol{ipk}, \Sigma, m, f_{\text{ev}})}$

$(opk, preosk = dk) \leftarrow osk; (ek, \cdot) \leftarrow opk$
if $\text{Verify}(opk, \boldsymbol{ipk}, \Sigma, m, f_{\text{ev}}) = 0$: **return** $\perp$
$((\pi_{\text{ev}}, c_{\text{op}}), y_{\text{ev}}) \leftarrow \Sigma$
$y_{\text{op}} \leftarrow \text{E.Dec}(dk, c_{\text{op}})$
$\pi_{\text{op}} \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{op}}}(crs_{\text{op}}, (ek, c, y_{\text{op}}), (dk))$
return $(y_{\text{op}}, \pi_{\text{op}})$

$\underline{\text{Judge}(opk, y_{\text{op}}, \pi_{\text{op}}, \Sigma, m)}$

if $\text{Verify}(opk, \boldsymbol{ipk}, \Sigma, m, f_{\text{ev}}) = 0$: **return** 0
$((\cdot, c_{\text{op}}), \cdot) \leftarrow \Sigma; (ek, \cdot) \leftarrow opk$
return $\text{NIZK.Verify}^{\mathcal{R}_{\text{op}}}(crs_{\text{op}}, \pi_{\text{op}}, (ek, c, y_{\text{op}}))$

Figure 3: Π_{UAS} algorithms 1/2: everything except issuing.

of 0. This is justified by IND-CPA security. Notably simulation extraction is needed to simulate decryption. For issuing anonymity we additionally assign all challenge credentials to the same virtual user. This is justified by the blinding property of the SBCM scheme.

Theorem 3 (Issuance unforgeability of Π_{UAS})**.** *If the* NIZK *scheme is extraction zero-knowledge and simulation extractable, and the* SBCM *scheme is correct, unforgeable, and has deterministically derived public keys, then* Π_{UAS} *satisfies issuance unforgeability as defined in Definition 4.*

Theorem 4 (Signature unforgeability of Π_{UAS})**.** *If the underlying NIZK scheme is complete, extraction zero-knowledge and simulation extractable, the public key encryption scheme is correct, and the* SBCM *scheme is correct, unforgeable, and has deterministically derived public keys, then* Π_{UAS} *satisfies signing unforgeability as defined in Definition 5.*

Theorem 5 (Non-frameability of Π_{UAS})**.** *If the NIZK system is extraction zero-knowledge and simulation extractable and the* SBCM *scheme is correct, blind, and unforgeable, then* Π_{UAS} *satisfies non-frameability as defined in Definition 6.*

The three unforgeability and non-frameability proofs share the intuition: Using simulation, we ensure via a series of game hops that we reach a game where we can embed an SBCM unforgeability

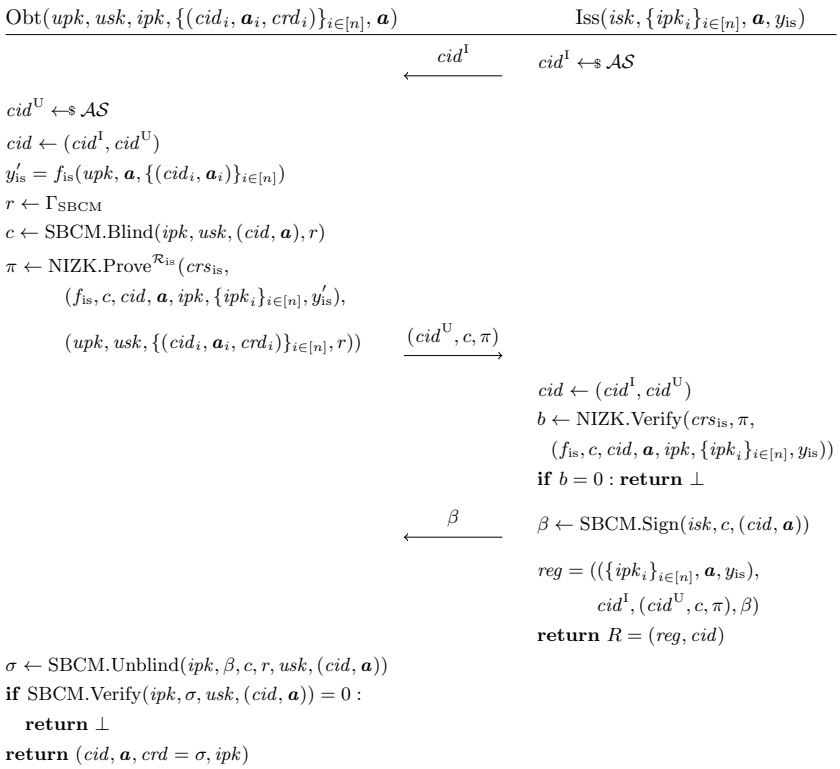

Figure 4: Π_{UAS} algorithms 2/2: issuing protocol. $\mathcal{AS}$ is an assumed attribute space.

challenge. This requires that the SBCM secret key is only used in operations that can be simulated using SBCM challenge oracles. Then, if the adversary wins the UAS game, we leverage its output to break SBCM unforgeability. Here, the main complexity is in alternating extraction and simulation: as our initial games already extract, we rely on extraction zero-knowledge NIZKs. Moreover, we need to ensure that we only extract from non-simulated proofs (and never attempt extraction of simulated ones).

6 Building Related Schemes from UAS

Π_{UAS} restrictions. Given a generic UAS construction, Π_{UAS}, we can restrict the achieved privacy-vs-utility tradeoff by requiring it to use concrete f^a_{is}, f^b_{ev} and f^c_{op} functions. We refer to the result as the $(f^a_{\text{is}}, f^b_{\text{ev}}, f^c_{\text{op}})$-$\Pi_{\mathsf{UAS}}$ *restriction*. Note that security of Π_{UAS} implies security of its restrictions.

To showcase the generality of UAS, we briefly describe concrete Π_{UAS} restrictions that instantiate vanilla digital signatures, group signatures, anonymous credentials, and ring signatures, using the functions defined in Fig. 5. Fig. 6 graphically depicts these connections. The instantiations based on our Π_{UAS} generic construction are probably not the most efficient approach to build the corresponding related scheme. Still, we see it as an initial feasibility result, from which to build more efficient instantiations – perhaps relying on alternative UAS constructions for more restricted

but still expressive enough function classes. We defer security models and proofs to Appendix D and show relations to other more recent variants of GS and AC schemes [EHK+19, NGSY22a] in Appendix E.

Issuance functions
$f_{\text{is}}^{\text{upk}}(upk, \cdot, \cdot) \coloneqq \textbf{return } upk$

Open functions
$f_{\text{op}}^{0}(\cdot, \cdot, \cdot) \coloneqq \textbf{return } 0$
$f_{\text{op}}^{\text{upk}}(upk, \cdot, \cdot) \coloneqq \textbf{return } upk$

Signature evaluation functions
$f_{\text{ev}}^{\text{upk}}(upk, \cdot, \cdot) \coloneqq \textbf{return } upk$
$f_{\text{ev}}^{0}(\cdot, \cdot, \cdot) \coloneqq \textbf{return } 0$
$f_{\text{ev}}^{\boldsymbol{d}}(\cdot, (\cdot, \boldsymbol{a}), \cdot) \coloneqq \textbf{return } (attr_i)_{i \in \boldsymbol{d}}$
$f_{\text{ev}}^{\boldsymbol{ring}}(upk, \cdot, \cdot) \coloneqq \textbf{if } upk \in \boldsymbol{ring} : \textbf{return } 1$
$\textbf{else return } 0$

Figure 5: Functions for the Π_{UAS} restrictions described next. "$\cdot$" denotes ignored arguments. $f^{\mathtt{a}}$ is a function named "a"; f^{a} is a function parameterized with a.

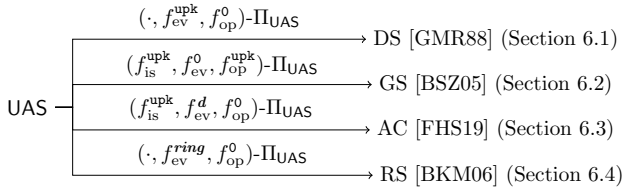

Figure 6: Π_{UAS}-restrictions instantiating related schemes.

6.1 Digital Signatures

As a warm up, we show that a $(\cdot, f_{\text{ev}}^{\text{upk}}, f_{\text{op}}^{0})$-$\Pi_{\mathsf{UAS}}$ restriction realizes a conventional digital signatures satisfying EUF-CMA security [GMR88]. No issuance function is required, as users do not need credentials to sign. The evaluation function outputs the signer's public key, and any opening function works – the output is ignored. A possible optimization would be using the empty string as ciphertext for constant functions (e.g. f_{op}^{0}) and skip verifiable encryption. Concretely, we create $\Pi_{\mathsf{UAS}}^{\text{ds}}$ from Π_{UAS}, where public parameters par are passed implicitly:

$\underline{\text{Setup}(1^\kappa)}$ $par' \leftarrow \Pi_{\mathsf{UAS}}.\text{Setup}(1^\kappa)$; $(preopk, preosk) \leftarrow \Pi_{\mathsf{UAS}}.\text{OKG}(par)$;
$opk \leftarrow (preopk, f_{\text{op}}^{0})$; $osk \leftarrow (opk, preosk)$; **return** $par = (par', opk)$.

$\underline{\text{KG}(1^\kappa)}$ **return** $(upk, usk) \leftarrow \Pi_{\mathsf{UAS}}.\text{KG}(par')$.

$\underline{\text{Sign}(usk, m)}$ $(\sigma, y_{\text{ev}}) \leftarrow \Pi_{\mathsf{UAS}}.\text{Sign}(upk, usk, opk, \emptyset, m, f_{\text{ev}}^{\text{upk}})$;
return σ. //$y_{\text{ev}} = upk$

$\underline{\text{Verify}(upk, m, \sigma)}$ **return** $\Pi_{\mathsf{UAS}}.\text{Verify}(opk, \emptyset, \Sigma = (\sigma, upk), m, f_{\text{ev}}^{\text{upk}})$.
//Since $f_{\text{ev}}^{\text{upk}}$ outputs the signer's upk, we know that σ is bound to the owner of upk.

6.2 Group Signatures

We show that a $(f_{\text{is}}^{\text{upk}}, f_{\text{ev}}^{0}, f_{\text{op}}^{\text{upk}})$-$\Pi_{\mathsf{UAS}}$ restriction is a secure group signature scheme in a definition in the spirit of [BSZ05]. First, the $f_{\text{is}}^{\text{upk}}$ function outputs the upk of the requesting user, allowing

the issuer to detect a user requesting multiple credentials – note that, in vanilla group signatures, there is a single issuer and a single membership certificate per user. Thus, linkable issuance is the expected behavior. The evaluation function $f^{\mathsf{0}}_{\mathrm{ev}}$ outputs a constant value. Finally, the opening function $f^{\mathtt{upk}}_{\mathrm{op}}$ outputs the signer's *upk*, allowing the opener to identify the signer of any group signature. Concretely, we create $\Pi^{\mathrm{gs}}_{\mathsf{UAS}}$ from Π_{UAS} as follows:

$\underline{\mathrm{KG}(1^\kappa)}$ $par \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{Setup}(1^\kappa)$//implicit; $(preopk, preosk) \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{OKG}(par)$; $opk \leftarrow (preopk, f^{\mathtt{upk}}_{\mathrm{op}})$; $osk \leftarrow (opk, preosk)$; $(ipk', isk') \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{KG}(par)$;
$ipk \leftarrow (ipk', f^{\mathtt{upk}}_{\mathrm{is}})$; $isk \leftarrow (ipk, isk')$; **return** $(gpk = (ipk, opk), isk, osk)$.

$\underline{\mathrm{UKG}(1^\kappa)}$ **return** $(upk, usk) \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{KG}(par)$.

$\underline{\langle \mathrm{Obt}(usk, ipk), \mathrm{Iss}((isk, opk), upk)\rangle}$
$\langle C, R\rangle \leftarrow \Pi_{\mathsf{UAS}}.\langle \mathrm{Obt}(upk, usk, ipk, \emptyset, \emptyset), \mathrm{Iss}(isk, \emptyset, \emptyset, y_{\mathrm{is}}{=}upk)\rangle$.
return $\langle (usk, C), R\rangle$ //The credential is locally augmented with the user's secret key.

$\underline{\mathrm{Sign}(gpk, (usk, C), m)}$ $(ipk, opk) \leftarrow gpk$; $(\sigma, y_{\mathrm{ev}}) \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{Sign}(upk, usk, opk, C, m, f^{\mathsf{0}}_{\mathrm{ev}})$; **return** σ
//$y_{\mathrm{ev}} = 0$ is the constant output of $f^{\mathsf{0}}_{\mathrm{ev}}$.

$\underline{\mathrm{Verify}(gpk, \sigma, m)}$ $(ipk, opk) \leftarrow gpk$; **return** $\Pi_{\mathsf{UAS}}.\mathrm{Verify}(opk, ipk, (\sigma, 0), m, f^{\mathsf{0}}_{\mathrm{ev}})$.

$\underline{\mathrm{Open}(gpk{=}(ipk, opk), osk, \sigma, m)}$ **return** $\Pi_{\mathsf{UAS}}.\mathrm{Open}(osk, ipk, \sigma, 0, m, f^{\mathsf{0}}_{\mathrm{ev}})$.

$\underline{\mathrm{Judge}(gpk{=}(ipk, opk), \pi, upk, \sigma, m)}$ **return** $\Pi_{\mathsf{UAS}}.\mathrm{Judge}(opk, upk, \pi, (\sigma, 0), m)$.

6.3 Anonymous Credentials

We show how to build AC systems from UAS signatures. For concreteness, we use a $(f^{\mathtt{upk}}_{\mathrm{is}}, f^{\boldsymbol{d}}_{\mathrm{ev}}, f^{\mathsf{0}}_{\mathrm{op}})$-$\Pi_{\mathsf{UAS}}$ restriction which suffices for the AC scheme of Fuchsbauer et al. [FHS19] which does not have issuance anonymity and supports selective disclosure of attributes. Thus, our issuance function returns the user's public key as for GS and the evaluation function reveals the chosen subset of attributes.

To match the syntax of the target scheme, we implement a simple challenge response protocol via UAS signing. The evaluation function $f^{\boldsymbol{d}}_{\mathrm{ev}}$ returns subset $D = (attr_i)_{i\in\boldsymbol{d}}$ of the attributes. The open function $f^{\mathsf{0}}_{\mathrm{op}}$ reveals nothing.

$\underline{\mathrm{IssKeyGen}(1^\kappa)}$ $par' \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{Setup}(1^\kappa)$; $(preopk, preosk) \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{OKG}(par')$; $opk \leftarrow (preopk, f^{\mathsf{0}}_{\mathrm{op}})$; $osk \leftarrow (opk, preosk)$; $(ipk', isk') \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{KG}(par')$;
$par \leftarrow (par', opk)$ //kept implicit ; **return** $(ipk{=}(ipk', f^{\mathtt{upk}}_{\mathrm{is}})), isk{=}(ipk, isk'))$.

$\underline{\mathrm{UserKeyGen}(1^\kappa)}$ **return** $(upk, usk) \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{KG}(par')$.

$\underline{\langle \mathrm{Obt}(usk, ipk, \boldsymbol{a}), \mathrm{Iss}(isk, upk, \boldsymbol{a})\rangle}$
$\langle C_{\mathsf{UAS}}, R\rangle \leftarrow \Pi_{\mathsf{UAS}}.\langle \mathrm{Obt}(upk, usk, ipk, \boldsymbol{C}{=}\emptyset, \boldsymbol{a}), \mathrm{Iss}(isk, \boldsymbol{ipk}{=}\emptyset, \boldsymbol{a}, y_{\mathrm{is}}{=}upk)\rangle$.
return $\langle C{=}(usk, C_{\mathsf{UAS}}), \mathbf{if}\ R \neq \bot : \top\rangle$//Local translation of protocol outputs.

$\underline{\mathrm{Show}(ipk, \boldsymbol{a}, \boldsymbol{d}, C{=}(usk, C_{\mathsf{UAS}})), \mathrm{Verify}(ipk, \boldsymbol{d}, D)\rangle}$
V: **send** $r \leftarrow \{0,1\}^\kappa$ **to** S
S: **send** $(\sigma, y_{\mathrm{ev}}) = \Sigma \leftarrow \Pi_{\mathsf{UAS}}.\mathrm{Sign}(upk, usk, opk, C_{\mathsf{UAS}}, r, f^{\boldsymbol{d}}_{\mathrm{ev}})$ **to** V
V: **return** $y_{\mathrm{ev}} = D \wedge \Pi_{\mathsf{UAS}}.\mathrm{Verify}(opk, ipk, \Sigma, r, f^{\boldsymbol{d}}_{\mathrm{ev}})$

6.4 Ring Signatures

We use a $(\cdot, f_{\text{ev}}^{\boldsymbol{ring}}, f_{\text{op}}^{0})$-$\Pi_{\mathsf{UAS}}$ restriction to build a ring signature scheme, $\Pi_{\mathsf{UAS}}^{\text{ring}}$. For signing, signers compute a $f_{\text{ev}}^{\boldsymbol{ring}}$ function, where $\boldsymbol{ring} = \{upk_i\}_{i\in[n]}$ is an arbitrary set of public keys. This function returns 1 if the signer's $upk \in \boldsymbol{ring}$, and 0 otherwise. A ring signature is a Π_{UAS} signature evaluated on such a $f_{\text{ev}}^{\boldsymbol{ring}}$ function – which does not require any credential. The construction is as follows:

$\underline{\text{Setup}(1^\kappa)}$ $par' \leftarrow \Pi_{\mathsf{UAS}}.\text{Setup}(1^\kappa)$; $(preopk, preosk) \leftarrow \Pi_{\mathsf{UAS}}.\text{OKG}(par)$;
$opk \leftarrow (preopk, f_{\text{op}}^{0})$; $osk \leftarrow (opk, preosk)$; **return** $par = (par', opk)$.
//We assume that the setup is trusted to compute the correct opk.

$\underline{\text{KG}(1^\kappa)}$ **return** $(pk, sk) \leftarrow \Pi_{\mathsf{UAS}}.\text{KG}(par')$.

$\underline{\text{Sign}(usk, \boldsymbol{ring}, m)}$ $(\sigma, y_{\text{ev}}) \leftarrow \Pi_{\mathsf{UAS}}.\text{Sign}(upk, usk, opk, \emptyset, m, f_{\text{ev}}^{\boldsymbol{ring}})$; **return** σ $//y_{\text{ev}} = (upk \in \boldsymbol{ring})$.

$\underline{\text{Verify}(\boldsymbol{ring}, m, \sigma)}$ **return** $\Pi_{\mathsf{UAS}}.\text{Verify}(opk, \emptyset, \Sigma = (\sigma, y_{\text{ev}} = 1), m, f_{\text{ev}}^{\boldsymbol{ring}})$.
//Since $f_{\text{ev}}^{\boldsymbol{ring}}$ outputs 1 we know that the signer is in the ring.

Note that the issuance function is never used, as no credential takes part in the signing process, so we simply ignore it. The same does not apply to the open function, though. Even if no actual Open function is exposed by the ring signature construction, a malicious party could try to open a signature. Thus, we need to fix it to a function that does not leak information, like f_{op}^{0}.

7 Conclusion and Future Work

We present a general model and construction for anonymous signatures, allowing for different privacy-vs-utility trade-offs. The flexibility of our model stems from functional placeholders that modulate the utility information learned by issuers, verifiers, and openers at credential issuance and authentication time, as well as after authentication. To showcase its generality, we show how to securely instantiate well-known schemes using our construction.

A further natural generalization of our model would allow for issuers and openers to adjust their functions dynamically, or to allow for multiple openers for the same signature. A practical concern are optimized implementations of Π_{UAS} restrictions and optimized constructions for restricted function classes. For example, the instantiation from the building blocks in Appendix B, based on BBS+, ElGamal, and basic sigma proofs, is well suited (and efficient) for cases that need selective disclosure. However, it falls short (or would be inefficient) for others.

References

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2000. doi:10.1007/3-540-44598-6_17.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, pages 111–125, 2006.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2009. doi:10.1007/978-3-642-03356-8_7.

[BCC+15] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on DDH. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, volume 9326 of *Lecture Notes in Computer Science*, pages 243–265. Springer, 2015. doi:10.1007/978-3-319-24174-6_13.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve A. Schneider, editors, *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, volume 9696 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2016. doi:10.1007/978-3-319-39555-5_7.

[BCD+17] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 301–315. IEEE, 2017. URL: https://doi.org/10.1109/EuroSP.2017.13, doi:10.1109/EUROSP.2017.13.

[BCN+10] Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Get shorty via group signatures without encryption. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, pages 381–398, 2010.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2006. doi:10.1007/11681878_4.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT 2003, Proceedings*, pages 614–629, 2003.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA 2005, Proceedings*, pages 136–153, 2005.

[CDL+13] Jan Camenisch, Maria Dubovitskaya, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. Concepts and languages for privacy-preserving attribute-based authentication. In Simone Fischer-Hübner, Elisabeth de Leeuw, and Chris J. Mitchell, editors, *Policies and Research in Identity Management - Third IFIP*

WG 11.6 Working Conference, IDMAN 2013, London, UK, April 8-9, 2013. Proceedings, volume 396 of *IFIP Advances in Information and Communication Technology*, pages 34–52. Springer, 2013. doi:10.1007/978-3-642-37282-7_4.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *TRUST 2016, Proceedings*, pages 1–20, 2016.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985. doi:10.1145/4372.4373.

[CKL+15] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal treatment of privacy-enhancing credential systems. In *SAC 2015, Revised Selected Papers*, pages 3–24, 2015.

[CKLM14] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 199–213. IEEE Computer Society, 2014. doi:10.1109/CSF.2014.22.

[CKS10] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. Solving revocation with efficient update of anonymous credentials. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 454–471. Springer, 2010. doi:10.1007/978-3-642-15317-4_28.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 2001, Proceeding*, pages 93–118, 2001.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002. doi:10.1007/3-540-36413-7_20.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2006. doi:10.1007/11818175_5.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997. doi:10.1007/BFb0052252.

[CvH91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT '91, Proceedings*, pages 257–265. Springer, 1991.

[DL21] Jesus Diaz and Anja Lehmann. Group signatures with user-controlled and sequential linkability. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 360–388. Springer, 2021. doi:10.1007/978-3-030-75245-3_14.

[DMM+18] Dominic Deuber, Matteo Maffei, Giulio Malavolta, Max Rabkin, Dominique Schröder, and Mark Simkin. Functional credentials. *Proc. Priv. Enhancing Technol.*, 2018(2):64–84, 2018. doi:10.1515/popets-2018-0013.

[EHK+19] Keita Emura, Goichiro Hanaoka, Yutaka Kawai, Takahiro Matsuda, Kazuma Ohara, Kazumasa Omote, and Yusuke Sakai. Group signatures with message-dependent opening: Formal definitions and constructions. *Secur. Commun. Networks*, 2019:4872403:1–4872403:36, 2019. doi:10.1155/2019/4872403.

[Esp22] Espresso Systems. Configurable Asset Privacy. https://github.com/EspressoSystems/cap/blob/main/cap-specification.pdf, 2022.

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *J. Cryptol.*, 32(2):498–546, 2019. doi:10.1007/s00145-018-9281-4.

[GL19] Lydia Garms and Anja Lehmann. Group signatures with selective linkability. In *PKC 2019, Proceedings*, pages 190–220, 2019.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. doi:10.1137/0217017.

[GO14] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *J. Cryptol.*, 27(3):506–543, 2014. doi:10.1007/s00145-013-9152-y.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006. doi:10.1007/11761679_21.

[KKS22] Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1739–1752. ACM, 2022. doi:10.1145/3548606.3560707.

[KLN23] Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. Privacy-preserving blueprints. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 594–625. Springer, 2023. doi:10.1007/978-3-031-30617-4_20.

[KY06] Aggelos Kiayias and Moti Yung. Secure scalable group signature with dynamic joins and separable authorities. *Int. J. Secur. Networks*, 1(1/2):24–45, 2006. doi:10.1504/IJSN.2006.010821.

[LNPY21] Benoît Libert, Khoa Nguyen, Thomas Peters, and Moti Yung. Bifurcated signatures: Folding the accountability vs. anonymity dilemma into a single private signing scheme. In Anne Canteaut and Franccois-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 521–552. Springer, 2021. doi:10.1007/978-3-030-77883-5_18.

[MSS06] Mark Manulis, Ahmad-Reza Sadeghi, and Jörg Schwenk. Linkable democratic group signatures. In *Information Security Practice and Experience, Second International Conference, ISPEC 2006, Hangzhou, China, April 11-14, 2006, Proceedings*, pages 187–201, 2006.

[NGSY22a] Khoa Nguyen, Fuchun Guo, Willy Susilo, and Guomin Yang. Multimodal private signatures. Cryptology ePrint Archive, Paper 2022/1008, 2022. https://eprint.iacr.org/2022/1008. URL: https://eprint.iacr.org/2022/1008.

[NGSY22b] Khoa Nguyen, Fuchun Guo, Willy Susilo, and Guomin Yang. Multimodal private signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 792–822. Springer, 2022. doi:10.1007/978-3-031-15979-4_27.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 427–437. ACM, 1990. doi:10.1145/100216.100273.

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 1996. doi:10.1007/BFb0034852.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA*

Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, volume 9610 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2016. doi:10.1007/978-3-319-29485-8_7.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer, 2001. doi:10.1007/3-540-44647-8_33.

[SEH+12] Yusuke Sakai, Keita Emura, Goichiro Hanaoka, Yutaka Kawai, Takahiro Matsuda, and Kazumasa Omote. Group signatures with message-dependent opening. In Michel Abdalla and Tanja Lange, editors, *Pairing-Based Cryptography - Pairing 2012 - 5th International Conference, Cologne, Germany, May 16-18, 2012, Revised Selected Papers*, volume 7708 of *Lecture Notes in Computer Science*, pages 270–294. Springer, 2012. doi:10.1007/978-3-642-36334-4_18.

[TZ23] Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 691–721. Springer, 2023. doi:10.1007/978-3-031-30589-4_24.

[WG23] Anoncreds WG. Hyperledger anoncreds. https://www.hyperledger.org/use/anoncreds, May 2023.

A Formal UAS Model

A.1 Global Variables and Oracles

Global variables. Users are referred to with user identifiers, uid; for credentials, we use *cid*; and for openers, oid. Although issuers are "extended" users, we use iid to refer to a user acting as issuer. For notational convenience, we often use these identifiers instead of the corresponding user/issuer/opener keys. All tables defined in Table 1 are initialized empty. Among them, it is worth emphasizing:

CRD. Indexable by *cid*, it stores $(\mathsf{uid}, crd, \mathsf{iid}, \boldsymbol{a}, n, \{(cid_i, \mathsf{iid}_i)\}_{i\in[n]})$ tuples, where uid identifies the credential owner, *crd* (if available) is the credential, iid is the issuer identifier, $\boldsymbol{a}$ are the attributes in *crd*, n is the number of credentials used to support this request, and (cid_i, iid_i) for $i \in [n]$ are the corresponding credential and issuer identifiers. We may use $\mathsf{CRD}[\boldsymbol{cid}]$ to refer to $\mathsf{CRD}[cid]$ for all $cid \in \boldsymbol{cid}$. Also, we sometimes use $\mathsf{CRD}[cid]$ to mean $C = (cid, \boldsymbol{a}, crd, \mathsf{IPK}[\mathsf{iid}])$ such that $\mathsf{CRD}[cid] = (\cdot, crd, \mathsf{iid}, \boldsymbol{a}, \cdot, \cdot)$. We may also write $\mathsf{CRD}[\boldsymbol{cid}] = \boldsymbol{C} = \{\mathsf{CRD}[cid_i]\}_{i\in[n]}$ for vectors of IDs. Finally, we write $\mathsf{CRD}[\mathsf{uid}]$ to denote the set of credentials of user uid, i.e. $\mathsf{CRD}[\mathsf{uid}] = \{(cid, \boldsymbol{a}, crd, \mathsf{IPK}[\mathsf{iid}]) \mid (\mathsf{uid}, crd, \mathsf{iid}, \boldsymbol{a}, \cdot, \cdot) \in \mathsf{CRD}\}$.

CCRD. Like CRD, but tracks challenge credentials. Contains entries of the form $(\mathsf{uid}_b^*, crd, \mathsf{iid}, \boldsymbol{a}, n, \{(cid_i^*, \mathsf{iid}_i)\}_{i\in[n]}, \mathsf{uid}_{1-b}^*)$, storing both uid_b and uid_{1-b}.

SIG. Signatures generated by SIGN: $(\mathsf{uid}, \mathsf{oid}, m, f_{\text{ev}}, \Sigma)$ tuples, where uid is the signer, oid is the opener, m is the signed message, f_{ev} is the used evaluation function, and $\Sigma = (\sigma, y_{\text{ev}})$ is the generated signature.

CSIG. Stores challenge signatures output to $\mathcal{A}$ in the signature anonymity game that depend on the hidden bit b (either directly or because it depends on credentials that depend on b). Like SIG, each entry is of the form $(\mathsf{uid}, \mathsf{oid}, m, f_{\text{ev}}, \Sigma)$.

For convenience, we allow indexing OWN and ISR by identifier sets (i.e. OWN[$\boldsymbol{cid}$], ISR[$\boldsymbol{cid}$]). To avoid inconsistencies, if OWN (resp. ISR) is called with a set $\boldsymbol{cid}$ including cids owned (resp. issued) by different uids (resp. iids), the output is $\perp$. Also, ATT[$\boldsymbol{cid}$] returns the union of all the attributes in $\boldsymbol{cid}$.

Table	Indexed by	Content
H{U, I, O}	uid,iid,oid	Honest users, issuers, and openers.
C{U, I, O}	uid,iid,oid	Corrupt users, issuers, and openers.
{U, I, O}K	uid,iid,oid	User/Issuer/Opener key pairs.
{U, I, O}PK	uid,iid,oid	User/Issuer/Opener public keys.
{U, I, O}SK	uid,iid,oid	User/Issuer/Opener private keys.
OWN	cid	uid of the owner of the credential identified by cid.
ATT	cid	Attribute set encoded in the credential identified by cid.
ISR	cid	iid of the issuer of the credential identified by cid.
SIG	/	Honest sigs: $(\mathsf{uid}, \mathsf{oid}, m, f_{\text{ev}}, \Sigma = (\sigma, y_{\text{ev}}))$.
CSIG	/	Chall sigs: $(\mathsf{uid}, \mathsf{oid}, m, f_{\text{ev}}, \Sigma = (\sigma, y_{\text{ev}}))$.
CRD	cid	Issued creds: $(\mathsf{uid}, crd, \mathsf{iid}, \boldsymbol{a}, n, (\boldsymbol{cid}, \mathsf{ISR}[\boldsymbol{cid}]))$.
CCRD	cid	Chall creds: $(\mathsf{uid}_b^*, crd, \mathsf{iid}, \boldsymbol{a}, n, (\boldsymbol{cid}_b, \mathsf{ISR}[\boldsymbol{cid}_b]), \mathsf{uid}_{1-b}^*)$.

Table 1: Global state tables. Details of SIG, CSIG, CRD, and CCRD in the text.

Oracles. We define several oracles to model adversarial capabilities. We use the conventional oracles for generating users or openers ({HU, HO}GEN for honest ones; {CU, CO}GEN for fully corrupt ones), or corrupting existing users, issuers, or openers ({U, I, O}CORR). Also, we use an ISET oracle for "upgrading" a user to an issuer. Importantly, note that the OCORR oracle, for corrupting openers, rejects queries if the corresponding opener's key pair has been used to produce challenge signatures. This set of oracles is formally defined in Fig. 7.

The oracles for obtaining (non-challenge) credentials are OBTISS, ISSUE, and OBTAIN. In OBTISS, both issuer and user are honest; in ISSUE, only the issuer is honest; in OBTAIN, only the user is honest. In these oracles, we essentially do some bookkeeping of all the information that honest participants see. For issuing challenge credentials, in the issuance anonymity game, we introduce the $\mathsf{OBTCHAL}_b$ oracle. To call this oracle, $\mathcal{A}$ specifies two challenge users and two sets of endorsement credentials – as well as one common issuer and set of attributes for the requested credentials. The challenge bit b defines which set of challenge user and credentials will be used. In this case, we need to prevent some trivial attacks: namely, no challenge and non-challenge credentials can be mixed in the same request, and the output of f_{is} has to be the same for both sets of challenge credentials.

To create signatures and operate on them, the adversary can use the SIGN and OPEN oracles. Besides some bookkeeping, the SIGN oracle prevents trivial attacks by disallowing using both challenge and non-challenge credentials to produce a signature. In queries involving (only) multiple

challenge credentials, we also make sure that, in their respective issuance protocols, all were paired with the same uid^*_{1-b}, as that would trivially leak b too. In addition, f_{ev} must output the same value for both challenge sets, and if the opener is corrupt, so does f_{op}. With respect to OPEN, we simply prevent opening challenge signatures. Finally, the $\mathsf{SIGCHAL}_b$ oracle, used in the signature anonymity game, given two sets of user and credentials, picks the one defined by the bit b, and produces a challenge signatures. The checks against trivial wins are equivalent to those of SIGN. Note though, that the $\mathsf{SIGCHAL}_b$ oracle is never used in conjunction with $\mathsf{OBTCHAL}_b$, so we do not need to prevent trivial wins involving challenge credentials.

The non-challenge oracles are defined in Fig. 8, and Fig. 9 defines the challenge ones.

$\mathsf{HUGEN}(\mathsf{uid})$

//Generate honest user
if $\mathsf{uid} \in \mathsf{HU} \vee \mathsf{uid} \in \mathsf{CU}$: **return** $\bot$
$(upk, usk) \leftarrow \text{KG}(par)$
$\mathsf{UK}[\mathsf{uid}] \leftarrow (upk, usk)$
$\mathsf{HU} \leftarrow \mathsf{HU} \cup \{\mathsf{uid}\}$
return upk

$\mathsf{CUGEN}(\mathsf{uid}, upk)$

//Create corrupted user
if $\mathsf{uid} \in \mathsf{HU} \cup \mathsf{CU}$: **return** $\bot$
$\mathsf{CU} \leftarrow \mathsf{CU} \cup \{\mathsf{uid}\}$
$\mathsf{UK}[\mathsf{uid}] = (upk, \bot)$
return $\top$

$\mathsf{UCORR}(\mathsf{uid})$

//Corrupt existing honest user
if $\mathsf{uid} \in \mathsf{CU} \vee \mathsf{uid} \notin \mathsf{HU}$: **return** $\bot$
if $\mathsf{uid} \in \mathsf{HI} \cup \mathsf{CI}$: **return** $\bot$
$\mathsf{HU} \leftarrow \mathsf{HU} \setminus \{\mathsf{uid}\}$
$\mathsf{CU} \leftarrow \mathsf{CU} \cup \{\mathsf{uid}\}$
return $(\mathsf{USK}[\mathsf{uid}], \mathsf{CRD}[\mathsf{uid}])$

$\mathsf{ISET}(\mathsf{uid}, f_{\text{is}})$

//Upgrade user to issuer
if $\mathsf{uid} \notin \mathsf{HU} \cup \mathsf{CU}$: **return** $\bot$
if $\mathsf{uid} \in \mathsf{HI} \cup \mathsf{CI}$: **return** $\bot$
if $f_{\text{is}} \notin \mathcal{F}_{\text{is}}$: **return** $\bot$
$\mathsf{IK}[\mathsf{uid}] \leftarrow (ipk = (\mathsf{UPK}[\mathsf{uid}], f_{\text{is}}), \mathsf{USK}[\mathsf{uid}])$
if $\mathsf{uid} \in \mathsf{HU}$: $\mathsf{HI} \leftarrow \mathsf{HI} \cup \{\mathsf{uid}\}$
if $\mathsf{uid} \in \mathsf{CU}$: $\mathsf{CI} \leftarrow \mathsf{CI} \cup \{\mathsf{uid}\}$
return ipk

$\mathsf{HOGEN}(\mathsf{oid}, f_{\text{op}})$

//Generate honest opener
if $\mathsf{oid} \in \mathsf{HO} \vee \mathsf{oid} \in \mathsf{CO}$: **return** $\bot$
if $f_{\text{op}} \notin \mathcal{F}_{\text{op}}$: **return** $\bot$
$(preopk, preosk) \leftarrow \text{OKG}(par)$
$\mathsf{OK}[\mathsf{oid}] \leftarrow (opk = (preopk, f_{\text{op}}), preosk)$
$\mathsf{HO} \leftarrow \mathsf{HO} \cup \{\mathsf{oid}\}$
return opk

$\mathsf{COGEN}(\mathsf{oid}, preopk, f_{\text{op}})$

//Create corrupted opener
if $\mathsf{oid} \in \mathsf{HO} \cup \mathsf{CO}$: **return** $\bot$
if $f_{\text{op}} \notin \mathcal{F}_{\text{op}}$: **return** $\bot$
$\mathsf{CO} \leftarrow \mathsf{CO} \cup \{\mathsf{oid}\}$
$\mathsf{OK}[\mathsf{oid}] = ((preopk, f_{\text{op}}), \bot)$
return $\top$

$\mathsf{OCORR}(\mathsf{oid})$

//Corrupt existing honest opener
if $\mathsf{oid} \in \mathsf{CO} \vee \mathsf{oid} \notin \mathsf{HO}$: **return** $\bot$
//Nontriviality: involved in chall sig?
if $\exists(\cdot, \mathsf{oid}, \cdot, \cdot, \cdot) \in \mathsf{CSIG}$: **return** $\bot$
$\mathsf{HO} \leftarrow \mathsf{HO} \setminus \{\mathsf{oid}\}$
$\mathsf{CO} \leftarrow \mathsf{CO} \cup \{\mathsf{oid}\}$
return $\mathsf{OSK}[\mathsf{oid}]$

$\mathsf{ICORR}(\mathsf{uid})$

//Corrupt existing honest issuer
if $\mathsf{uid} \notin \mathsf{HI}$: **return** $\bot$
$\mathsf{HI} \leftarrow \mathsf{HI} \setminus \{\mathsf{uid}\}$
$\mathsf{CI} \leftarrow \mathsf{CI} \cup \{\mathsf{uid}\}$
$\mathsf{HU} \leftarrow \mathsf{HU} \setminus \{\mathsf{uid}\}$
$\mathsf{CU} \leftarrow \mathsf{CU} \cup \{\mathsf{uid}\}$
return $(\mathsf{USK}[\mathsf{uid}], \mathsf{CRD}[\mathsf{uid}])$

Figure 7: Detailed oracles available in our model (1/3). Oracles for key generation.

OBTISS($\mathsf{uid}, \mathsf{iid}, \boldsymbol{a}, \boldsymbol{cid}$)

//Honest issuer issues to honest user
//Require: both honest & no challenge creds
if $\mathsf{uid} \notin \mathsf{HU} \vee \mathsf{iid} \notin \mathsf{HI}$: **return** $\perp$
if $\exists i \in [n]$ *s.t.* $cid_i \in \mathsf{CCRD}$: **return** $\perp$
//Run issue/obtain protocol
$(\cdot, f_{\mathrm{is}}) \leftarrow \mathsf{IPK}[\mathsf{iid}]$
$y_{\mathrm{is}} \leftarrow f_{\mathrm{is}}(\mathsf{UPK}[\mathsf{uid}], \boldsymbol{a}, (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]))$
$\langle C, R\rangle \leftarrow \langle \mathrm{Obt}(\mathsf{UPK}[\mathsf{uid}], \mathsf{USK}[\mathsf{uid}], \mathsf{CRD}[\boldsymbol{cid}], \boldsymbol{a}),$
$\quad \mathrm{Iss}(\mathsf{ISK}[\mathsf{iid}], \mathsf{ISR}[\boldsymbol{cid}], \boldsymbol{a}, y_{\mathrm{is}})\rangle$
$(reg, cid) \leftarrow R$
//Bookkeeping: register this issuance
if $cid \in \mathsf{CRD} \vee cid \in \mathsf{CCRD}$: $coll \leftarrow 1$; **return** $\perp$
$\mathbf{reg}[cid] \leftarrow (y_{\mathrm{is}}, reg)$
$\mathsf{CRD}[cid] \leftarrow (\mathsf{uid}, crd, \mathsf{iid}, \boldsymbol{a}, n, (\boldsymbol{cid}, \mathsf{ISR}[\boldsymbol{cid}]))$
return $\top$

ISSUE($\mathsf{iid}, \boldsymbol{a}, \mathbf{iid} = \{\mathsf{iid}_i\}_{i\in[n]}, y_{\mathrm{is}}$)

//Honest issuer issues to $\mathcal{A}$
//Require: honest issuer
if $\mathsf{iid} \notin \mathsf{HI}$: **return** $\perp$
//Issue to $\mathcal{A}$
$\langle \cdot, R\rangle \leftarrow \langle \mathcal{A}, \mathrm{Iss}(\mathsf{ISK}[\mathsf{iid}], \mathbf{iid}, \boldsymbol{a}, y_{\mathrm{is}})\rangle$
if $R = \perp$: **return** $\perp$
$(reg, cid) \leftarrow R$
//Bookkeeping: register this issuance
if $cid \in \mathsf{CRD}$: $coll \leftarrow 1$; **return** $\perp$
$\mathbf{reg}[cid] \leftarrow (y_{\mathrm{is}}, reg)$
$\mathsf{CRD}[cid] \leftarrow (\perp, \perp, \mathsf{iid}, \boldsymbol{a}, n, \{(\perp, \mathsf{iid}_i)\}_{i\in[n]})$
return $\top$

OBTAIN($\mathsf{uid}, \mathsf{iid}, \boldsymbol{a}, \boldsymbol{cid}$)

//$\mathcal{A}$ issues to honest user
//Require: honest user, no challenge credentials
if $\mathsf{uid} \notin \mathsf{HU} \vee \mathsf{iid} \notin \mathsf{CI}$: **return** $\perp$
if $\exists i \in [n]$ *s.t.* $cid_i \in \mathsf{CCRD}$: **return** $\perp$
//Obtain credential from $\mathcal{A}$
$\langle C, \cdot\rangle \leftarrow \langle \mathrm{Obt}(\mathsf{UPK}[\mathsf{uid}], \mathsf{USK}[\mathsf{uid}], \mathsf{IPK}[\mathsf{iid}],$
$\quad \mathsf{CRD}[\boldsymbol{cid}], \boldsymbol{a}), \mathcal{A}\rangle$
$(cid, \cdot, crd, \cdot) \leftarrow C$
//Bookkeeping: register received credential
if $cid \in \mathsf{CRD} \vee cid \in \mathsf{CCRD}$: $coll \leftarrow 1$; **return** $\perp$
$\mathsf{CRD}[cid] \leftarrow (\mathsf{uid}, crd, \mathsf{iid}, \boldsymbol{a}, n, (\boldsymbol{cid}, \mathsf{ISR}[\boldsymbol{cid}]))$
return $\top$

OPEN($\mathsf{oid}, \mathbf{iid}, \Sigma, m, f_{\mathrm{ev}}$)

//Opener reveals opening y_{op} for Σ
//Require: Σ is not challenge signature
if $(\cdot, \cdot, \cdot, \cdot, \Sigma) \in \mathsf{CSIG}$:
  return $\perp$
//Open Σ to receive y_{op} and proof π
$(y_{\mathrm{op}}, \pi) \leftarrow \mathrm{Open}(\mathsf{OSK}[\mathsf{oid}], \mathsf{IPK}[\mathbf{iid}], \Sigma, m, f_{\mathrm{ev}})$
return (y_{op}, π)

SIGN($\mathsf{oid}, \mathsf{uid}, \boldsymbol{cid} = \{cid_i\}_{i\in[n]}, m, f_{\mathrm{ev}}$)

//Honest user creates a signature
//If $\boldsymbol{cid}$s given: ignore uid arg, set to $\boldsymbol{cid}$ owner
if $n > 0$: set (overwrite) $\mathsf{uid} = \mathsf{OWN}[\boldsymbol{cid}]$
//Require: Honest user, no mixing challenge creds
if $\mathsf{uid} \notin \mathsf{HU} \vee f_{\mathrm{ev}} \notin \mathcal{F}_{\mathrm{ev}}$: **return** $\perp$
if $\exists i \neq j \in [n]$ *s.t.* $cid_i \in \mathsf{CRD} \wedge cid_j \in \mathsf{CCRD}$:
  return $\perp$
$y_{\mathrm{ev}} \leftarrow f_{\mathrm{ev}}(\mathsf{UPK}[\mathsf{uid}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]), m)$
//If challenge cred is involved: nontriviality checks
if $cid_1 \in \mathsf{CCRD}$:
  //Credential ownership does not trivially leak b
  for $i \in [n]$: $(\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{uid}^*_{1-b,i}) \leftarrow \mathsf{CCRD}[cid_i]$
  if $\exists i \neq j \in [n]$ *s.t.* $\mathsf{uid}^*_{1-b,i} \neq \mathsf{uid}^*_{1-b,j}$:
    return $\perp$
  if $\mathsf{uid}^*_{1-b,1} \notin \mathsf{HU}$: **return** $\perp$
  //f_{ev} does not trivially leak b
  if $f_{\mathrm{ev}}(\mathsf{UPK}[\mathsf{uid}^*_{1-b,1}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]), m) \neq y_{\mathrm{ev}}$:
    return $\perp$
  //f_{op} does not trivially leak b
  $((\cdot, f_{\mathrm{op}}), \cdot) \leftarrow \mathsf{OPK}[\mathsf{oid}]$
  if $\mathsf{oid} \in \mathsf{CO} \wedge$
    $f_{\mathrm{op}}(\mathsf{UPK}[\mathsf{uid}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]), m) \neq$
    $f_{\mathrm{op}}(\mathsf{UPK}[\mathsf{uid}^*_{1-b,1}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]), m)$:
    return $\perp$
//Create signature + bookkeeping
if $cid_1 \in \mathsf{CCRD}$:
  $\Sigma \leftarrow \mathrm{Sign}(\mathsf{UPK}[\mathsf{uid}], \mathsf{USK}[\mathsf{uid}], \mathsf{OPK}[\mathsf{oid}], \mathsf{CCRD}[\boldsymbol{cid}], m, f_{\mathrm{ev}})$
  $\mathsf{CSIG} \leftarrow \mathsf{CSIG} \cup \{(\mathsf{uid}, \mathsf{oid}, m, f_{\mathrm{ev}}, \Sigma)\}$
else :
  $\Sigma \leftarrow \mathrm{Sign}(\mathsf{UPK}[\mathsf{uid}], \mathsf{USK}[\mathsf{uid}], \mathsf{OPK}[\mathsf{oid}], \mathsf{CRD}[\boldsymbol{cid}], m, f_{\mathrm{ev}})$
  $\mathsf{SIG} \leftarrow \mathsf{SIG} \cup \{(\mathsf{uid}, \mathsf{oid}, m, f_{\mathrm{ev}}, \Sigma)\}$
return Σ

Figure 8: Detailed oracles available in our model (2/3). Oracles for obtaining credentials, signatures, and processing them, excluding oracles specific to the anonymity games.

Correctness. As usual, correctness ensures that an honestly produced signature is accepted by Verify, and an honestly produced (y_{op}, π) pair is accepted by Judge. We also check that the y_{ev} and

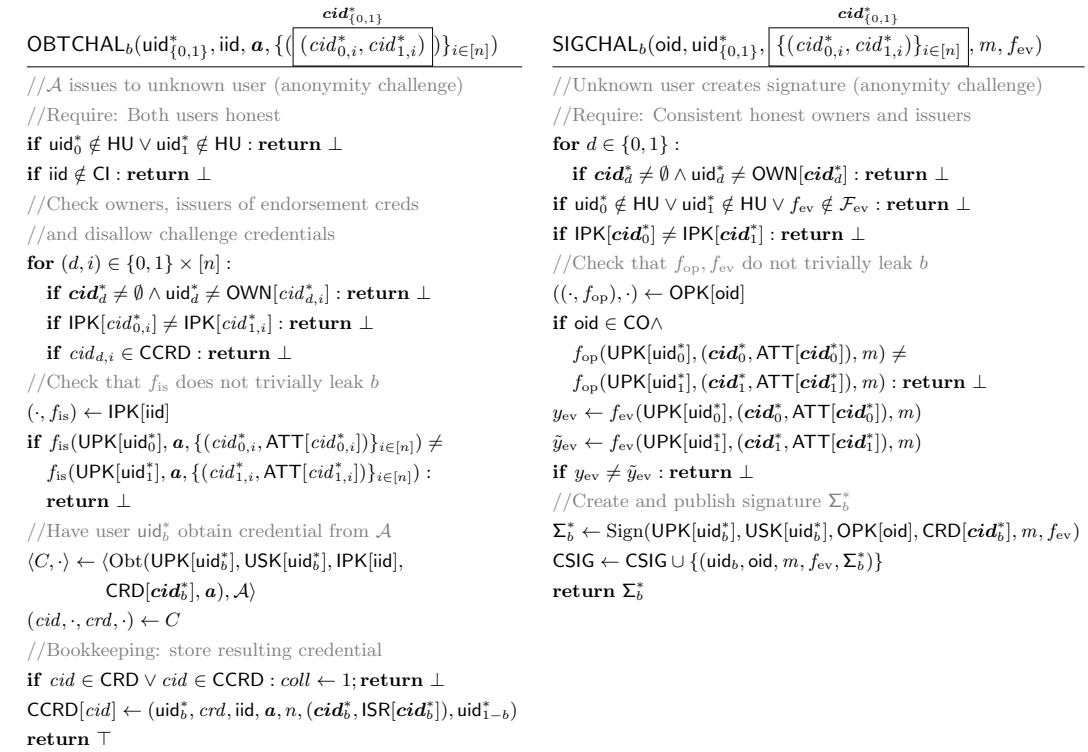

Figure 9: Detailed oracles available in our model (3/3). Oracles specific to the anonymity games.

y_{op} values are the result of a correct computation of f_{ev} and f_{op}, and that all credentials involved in the signature were issued correctly – in particular, that the y_{is} value provided during the issuance protocol matches the output of f_{is}. We emphasize that our correctness definition also enforces uniqueness in the credential identifiers. Concretely, even when the adversary *does not* follow the rules (e.g., outputs a fully corrupted oid), it can still win if it manages to produce two credentials with colliding identifiers (this is captured in the OBTISS, OBTAIN, ISSUE and OBTCHAL$_b$ oracles, which set the *coll* variable if that happens, even though they later abort issuance). For this, we additionally give the adversary access to all the oracles. We formally define correctness in Fig. 10. An UAS scheme is said to be correct as per Definition 1.

Definition 1. *(Correctness of* UAS*) A* UAS *scheme is correct if, for any p.p.t. adversary* $\mathcal{A}$*,* $\mathsf{Exp}^{\text{correct}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$ *outputs 1 with negligible probability.*

A.2 Security Properties

Helper functions. We require the existence of three helper functions – not available in the actual scheme, but rather to the challenger in the experiments. We introduce them here, and give concrete definitions for our construction, in Appendix C.

$\mathsf{Exp}^{\text{correct}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{Setup}(1^\kappa); coll \leftarrow 0$
$(\mathsf{uid}, \mathsf{oid}, \boldsymbol{cid}, m, f_{\text{ev}}) \leftarrow \mathcal{A}^{\mathcal{O}_{corr}}(par)$
//uid must be honest; *osk* must be known
//$\boldsymbol{cid}$ issuers must be honest
if $\mathsf{uid} \notin \mathsf{HU} \vee \mathsf{OSK}[\mathsf{oid}] = \perp \vee$
  $\exists cid \in \boldsymbol{cid}$ *s.t.* $\mathsf{ISR}[cid] \notin \mathsf{HI}$: **return** *coll*
//f_{ev} is valid; the same uid owns all creds
if $f_{\text{ev}} \notin \mathcal{F}_{\text{ev}} \vee \mathsf{OWN}[\boldsymbol{cid}] \neq \mathsf{uid}$: **return** *coll*
$\Sigma = (\sigma, y_{\text{ev}}) \leftarrow \text{Sign}(\mathsf{UPK}[\mathsf{uid}], \mathsf{USK}[\mathsf{uid}],$
  $\mathsf{OPK}[\mathsf{oid}], \mathsf{CRD}[\boldsymbol{cid}], m, f_{\text{ev}})$
//Honest verifiers accept honest sigs
if $\text{Verify}(\mathsf{OPK}[\mathsf{oid}], \mathsf{IPK}[\boldsymbol{cid}], \Sigma, m, f_{\text{ev}}) = 0$:
  return 1
//All endorsement credentials met their f_{is}
for $cid \in \boldsymbol{cid}$ **do** :
  $(\cdot, \cdot, \cdot, \boldsymbol{a}, n, \{(cid_i, \cdot)\}_{i\in[n]}) \leftarrow \mathsf{CRD}[cid]$
  $(y_{\text{is}}, reg) \leftarrow \mathbf{reg}[cid]; ((\cdot, f_{\text{is}}), \cdot) \leftarrow \mathsf{IPK}[cid]$
  if $f_{\text{is}}(\mathsf{UPK}[\mathsf{uid}], \boldsymbol{a}, (\{(cid_i, \mathsf{ATT}[cid_i])\}_{i\in[n]})) \neq y_{\text{is}}$:
  return 1
endfor
//y_{ev} matches an honestly computed f_{ev} output
$y'_{\text{ev}} \leftarrow f_{\text{ev}}(\mathsf{UPK}[\mathsf{uid}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]), m)$
if $y_{\text{ev}} \neq y'_{\text{ev}}$: **return** 1
$(y_{\text{op}}, \pi) \leftarrow \text{Open}(\mathsf{OSK}[\mathsf{oid}], \mathsf{IPK}[\boldsymbol{cid}], \Sigma, m, f_{\text{ev}})$
Parse $\mathsf{OPK}[\mathsf{oid}]$ as $((\cdot, f_{\text{op}}), \cdot)$
//Honest Judge accepts honest Open outputs,
//wherein y_{op} matches the value output by Open
if $\text{Judge}(\mathsf{OPK}[\mathsf{oid}], \mathsf{IPK}[\boldsymbol{cid}], y_{\text{op}}, \pi, \Sigma, m, f_{\text{ev}}) = 0 \vee$
  $y_{\text{op}} \neq f_{\text{op}}(\mathsf{UPK}[\mathsf{uid}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}]), m))$:
  return 1
return *coll*

Figure 10: Correctness experiment for UAS schemes. $\mathcal{O}_{corr}$ includes all oracles in Fig. 7, Fig. 8 and Fig. 9.

$\mathsf{SimSetup}(1^\kappa) \to (par, \tau)$**.** Given a security parameter, outputs global parameters *par* whose distribution is computationally indistinguishable to that produced by the Setup algorithm[2], as well as a trapdoor τ.

$\mathsf{ExtIss}(\tau, reg) \to (upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]})$**.** Receives trapdoor τ and a valid $\langle \text{Obt}, \text{Iss} \rangle$ transcript *reg*. It deterministically outputs the receiving user's public key *upk* and IDs and attributes of the credentials used by the user for the request.

$\mathsf{ExtSign}(\tau, \Sigma) \to (upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]})$**.** Receives trapdoor τ and a valid signature Σ. It deterministically outputs the signing user's public key *upk* and credential set $\boldsymbol{C}$ used to generate the signature.

These helpers are referenced by the oracles and experiments, and each security definition should be read as "is secure with respect to SimSetup, ExtIss, ExtSign". Schemes must define these helpers once, such that all security notions are fulfilled with respect to those three helpers.

Anonymity. The formal specification of the anonymity games is given in Fig. 11, and the definition of an issuance (resp. signature) anonymous UAS scheme in Definition 2 (resp. Definition 3).

Definition 2. *(Issuance anonymity in* UAS*) We define the advantage* $\mathsf{Adv}^{\text{iss-anon}}_{\mathsf{UAS},\mathcal{A}}$ *of* $\mathcal{A}$ *against* $\mathsf{Exp}^{\text{iss-anon-}b}_{\mathsf{UAS},\mathcal{A}}$ *as* $\mathsf{Adv}^{\text{iss-anon}}_{\mathsf{UAS},\mathcal{A}} = |\Pr[\mathsf{Exp}^{\text{iss-anon-1}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{iss-anon-0}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]|$*. A* UAS *scheme satisfies issuance anonymity if, for any p.p.t. adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\text{iss-anon}}_{\mathsf{UAS},\mathcal{A}}$ *is a negligible function of* 1^κ*.*

[2]One should consider this indistinguishability an implicit but important requirement of all security definitions that make use of SimSetup.

$\mathsf{Exp}^{\text{iss-anon-}b}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$

1 : $par \leftarrow \text{Setup}(1^\kappa)$
2 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\text{anon-b}},\mathsf{OBTCHAL}_b}(par)$
3 : **return** b^*

$\mathsf{Exp}^{\text{sig-anon-}b}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$

1 : $par \leftarrow \text{Setup}(1^\kappa)$
2 : $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\text{anon-b}},\mathsf{SIGCHAL}_b}(par)$
3 : **return** b^*

Figure 11: Issuance and signature anonymity experiments for UAS schemes. $\mathcal{O}_{\text{anon-b}} \leftarrow (\{\mathsf{HO},\mathsf{CO},\mathsf{HU},\mathsf{CU}\}\mathsf{GEN},\mathsf{ISET},\{\mathsf{I},\mathsf{O},\mathsf{U}\}\mathsf{CORR},\mathsf{ISSUE},\mathsf{OBTISS},\mathsf{OBTAIN},\mathsf{SIGN},\mathsf{OPEN})$.

Definition 3. *(Signature anonymity in* UAS*) We define the advantage* $\mathsf{Adv}^{\text{sig-anon}}_{\mathsf{UAS},\mathcal{A}}$ *of* $\mathcal{A}$ *against* $\mathsf{Exp}^{\text{sig-anon-}b}_{\mathsf{UAS},\mathcal{A}}$ *as* $\mathsf{Adv}^{\text{sig-anon}}_{\mathsf{UAS},\mathcal{A}} = |\Pr[\mathsf{Exp}^{\text{sig-anon-1}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{sig-anon-0}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]|$. *A* UAS *scheme satisfies signature anonymity if, for any p.p.t. adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{sig-anon}}_{\mathsf{UAS},\mathcal{A}}$ *is a negligible function of* 1^κ.

Unforgeability. The unforgeability-related experiments are given in Fig. 12, and the definition of an issuance unforgeable (resp. signature unforgeable) UAS scheme in Definition 4 (resp. Definition 5).

Definition 4. *(Unforgeable issuance of* UAS*) We define the advantage* $\mathsf{Adv}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$ *of* $\mathcal{A}$ *against* $\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$ *as* $\mathsf{Adv}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}} = \Pr[\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]$. *A* UAS *scheme has unforgeable issuance if, for any p.p.t. adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$ *is a negligible function of* 1^κ.

Definition 5. *(Unforgeable signatures of* UAS*) We define the advantage* $\mathsf{Adv}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ *of* $\mathcal{A}$ *against* $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ *as* $\mathsf{Adv}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}} = \Pr[\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]$. *A* UAS *scheme has unforgeable signing if, for any p.p.t. adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ *is a negligible function of* 1^κ.

Non-frameability. The non-frameability experiment is specified in Fig. 13, and the corresponding definition of non-frameable UAS schemes is given in Definition 6.

Definition 6. *(Non-frameability of* UAS*) We define the advantage* $\mathsf{Adv}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}$ *of* $\mathcal{A}$ *against* $\mathsf{Exp}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}$ *as* $\mathsf{Adv}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}} = \Pr[\mathsf{Exp}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]$. *A* UAS *scheme satisfies non-frameability if, for any p.p.t. adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}$ *is a negligible function of* 1^κ.

B Cryptographic Building Blocks

B.1 Public-Key Encryption

A public-key encryption scheme is defined by the following algorithms:

$par \leftarrow \text{E.Setup}(1^\kappa)$**.** Produces public parameters par given a security parameter 1^κ.

$(ek, dk) \leftarrow \text{E.KG}(par)$**.** Given public parameters par, produces an encryption-decryption key pair (ek, dk).

$c \leftarrow \text{E.Enc}(ek, m)$**.** Encrypts message m with encryption key ek, producing ciphertext c.

$m \leftarrow \text{E.Dec}(dk, c)$**.** Decrypts ciphertext c with decryption key dk. A deterministic algorithm.

$\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$

1 : //Set up with extraction trapdoor τ
2 : $(par, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$
3 : //$\mathcal{A}$ challenges extractor on issuance of cid
4 : $cid \leftarrow \mathcal{A}^{\mathcal{O}_{\text{forge}}}(par)$
5 : //Require: cid issued by honest issuer to $\mathcal{A}$.
6 : **if** $\mathsf{ISR}[cid] \notin \mathsf{HI} \vee \mathbf{reg}[cid] = \bot$: **return** 0
7 : **if** $\mathsf{OWN}[cid] \in \mathsf{HU}$: **return** 0
8 : //Retrieve & extract data on cid issuance
9 : $(\cdot, \cdot, \mathsf{iid}, \boldsymbol{a}, n, \{(\cdot, \mathsf{iid}_i)\}_{i\in[n]}) \leftarrow \mathsf{CRD}[cid]$
10 : $(ipk, f_{\text{is}}) \leftarrow \mathsf{IPK}[\mathsf{iid}], (y_{\text{is}}, reg) \leftarrow \mathbf{reg}[cid]$
11 : $(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}) \leftarrow \mathsf{ExtIss}(\tau, reg)$
12 : //$\mathcal{A}$ wins if extraction invalid for y_{is}
13 : **if** $f_{\text{is}}(upk, \boldsymbol{a}, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}) \neq y_{\text{is}}$: **return** 1
14 : //$\mathcal{A}$ wins if extracted creds not issued
15 : **return** $\mathsf{CheckEndorsementCreds}(upk, \{cid_i, \boldsymbol{a}_i, \mathsf{iid}_i\}_{i\in[n]})$

$\mathsf{CheckEndorsementCreds}(upk, \{cid_i, \boldsymbol{a}_i, \mathsf{iid}_i\}_{i\in[n]})$

1 : **for** $i \in \{i \in [n] : \mathsf{iid}_i \in \mathsf{HI}\}$:
2 : //$\mathcal{A}$ wins if honest issuer didn't issue $\boldsymbol{a}_i$
3 : **if** $\mathsf{ISR}[cid_i] \neq \mathsf{iid}_i \vee \mathsf{ATT}[cid_i] \neq \boldsymbol{a}_i$: **return** 1
4 : //$\mathcal{A}$ wins if cred belongs to honest user
5 : **if** $\mathsf{OWN}[cid_i] \in \mathsf{HU}$: **return** 1
6 : //Retrieve & extract data on issuance of cid_i
7 : $(\cdot, reg') \leftarrow \mathbf{reg}[cid_i], (upk', \cdot) \leftarrow \mathsf{ExtIss}(\tau, reg')$
8 : //$\mathcal{A}$ wins if cred was not issued to same upk
9 : **if** $upk \neq upk'$: **return** 1
10 : **return** 0

$\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$

1 : //Set up with extraction trapdoor τ
2 : $(par, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$
3 : //$\mathcal{A}$ outputs UAS signature forgery candidate Σ
4 : $(\mathsf{oid}, \mathbf{iid}, \Sigma = (\sigma, y_{\text{ev}}), m, f_{\text{ev}}) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{forge}}}(par)$
5 : //Require: signature was not produced by honest user
6 : **if** $(\cdot, \cdot, m, f_{\text{ev}}, \Sigma) \in \mathsf{SIG}$: **return** 0
7 : //Require: signature is accepted by honest Verify
8 : **if** $\text{Verify}(\mathsf{OPK}[\mathsf{oid}], \mathsf{IPK}[\mathbf{iid}], \Sigma, m, f_{\text{ev}}) = 0$: **return** 0
9 : //Use extractor to retrieve hidden values in Σ
10 : $(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}) \leftarrow \mathsf{ExtSign}(\tau, \Sigma)$
11 : //$\mathcal{A}$ wins if extraction inconsistent to y_{ev}
12 : **if** $f_{\text{ev}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \neq y_{\text{ev}}$: **return** 1
13 : //$\mathcal{A}$ wins if Σ cannot be opened correctly
14 : **if** $\mathsf{OSK}[\mathsf{oid}] \neq \bot$:
15 : $(y_{\text{op}}, \pi) \leftarrow \text{Open}(\mathsf{OSK}[\mathsf{oid}], \mathbf{iid}, \Sigma, m, f_{\text{ev}})$
16 : $(\cdot, f_{\text{op}}) \leftarrow \mathsf{OPK}[\mathsf{oid}]$
17 : **if** $\text{Judge}(\mathsf{OPK}[\mathsf{oid}], \mathsf{IPK}[\mathbf{iid}], y_{\text{op}}, \pi, \Sigma, m, f_{\text{ev}}) = 0 \vee$
18 : $f_{\text{op}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \neq y_{\text{op}}$: **return** 1
19 : //$\mathcal{A}$ wins if extracted creds not issued
20 : **return** $\mathsf{CheckEndorsementCreds}(upk, \{cid_i, \boldsymbol{a}_i, \mathsf{iid}_i\}_{i\in[n]})$

Figure 12: Unforgeability experiments in UAS schemes. $\mathcal{O}_{\text{forge}} \leftarrow \{\mathsf{HO}, \mathsf{CO}, \mathsf{HU}, \mathsf{CU}\}\mathsf{GEN}, \mathsf{ISET}, \{\mathsf{O}, \mathsf{U}, \mathsf{I}\}\mathsf{CORR}, \mathsf{OBTAIN}, \mathsf{OBTISS}, \mathsf{ISSUE}, \mathsf{SIGN}, \mathsf{OPEN}$.

A public-key encryption scheme is correct if, given a honestly generated key pair (ek, dk), produced with honestly generated parameters par_{E}, $\Pr[\text{E.Dec}(dk, \text{E.Enc}(ek, m)) = m] = 1$.

A public-key encryption scheme has IND-CPA security if $\Pr[\mathsf{Exp}^{\text{IND-CPA-1}}_{\text{E},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{IND-CPA-0}}_{\text{E},\mathcal{A}}(1^\kappa) = 1]|$ is a negligible function of κ_{E}, for any p.p.t. adversary $\mathcal{A}$, where $\mathsf{Exp}^{\text{IND-CPA-}b}_{\text{E},\mathcal{A}}$ is as defined in Fig. 14.

A public-key encryption scheme has IND-CCA security if $\Pr[\mathsf{Exp}^{\text{IND-CCA-1}}_{\text{E},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{IND-CCA-0}}_{\text{E},\mathcal{A}}(1^\kappa) = 1]|$ is a negligible function of κ_{E}, for any p.p.t. adversary $\mathcal{A}$, where $\mathsf{Exp}^{\text{IND-CCA-}b}_{\text{E},\mathcal{A}}$ is as defined in Fig. 14.

Proving that a key pair was correctly generated. In $\mathcal{R}_{\text{op}}$ of Π_{UAS}, we include statements requiring to prove that a key pair was produced by an E.KG algorithm (therein denoted $(opk, osk) \in [\text{OKG}(par, \cdot)]$). While how to do this highly depends on the concrete construction, note that it can be as trivial as proving knowledge of a discrete logarithm – as is the case in ElGamal encryption, which is IND-CPA secure under the DDH assumption, as required by UAS.

$\mathsf{Exp}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$

$(par, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$
$(\mathsf{oid}, \mathbf{iid}, \Sigma = (\sigma, y_{\text{ev}}), m, f_{\text{ev}}, y_{\text{op}}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{frame}}}(par)$
//Require: signature was not produced by honest user
if $(\cdot, \mathsf{oid}, m, f_{\text{ev}}, \Sigma) \in \mathsf{SIG}$: **return** 0
//Require: signature is accepted by honest Verify
if $\text{Verify}(\mathsf{OPK}[\mathsf{oid}], \mathsf{IPK}[\mathbf{iid}], \Sigma, m, f_{\text{ev}}) = 0$: **return** 0
//Require: opening proof is accepted by honest Judge
if $\text{Judge}(\mathsf{OPK}[\mathsf{oid}], \mathsf{IPK}[\mathbf{iid}], y_{\text{op}}, \pi, \Sigma, m) = 0$: **return** 0
$(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}) \leftarrow \mathsf{ExtSign}(\tau, \Sigma)$
//$\mathcal{A}$ wins if f_{op} evaluated on extracted values doesn't match y_{op}
$(\cdot, f_{\text{op}}) \leftarrow \mathsf{OPK}[\mathsf{oid}]$
if $f_{\text{op}}(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i\in[n]}, m) \neq y_{\text{op}}$: **return** 1
//$\mathcal{A}$ wins if the extracted upk belongs to an honest user
if $\exists \mathsf{uid} \in \mathsf{HU}$ *s.t.* $\mathsf{UPK}[\mathsf{uid}] = upk$: **return** 1
return 0

Figure 13: Experiment for non-frameability on UAS schemes. $\mathcal{O}_{\text{frame}} \leftarrow \{\mathsf{HO}, \mathsf{CO}, \mathsf{HU}, \mathsf{CU}\}\mathsf{GEN}, \mathsf{ISET}, \{\mathsf{I}, \mathsf{O}, \mathsf{U}\}\mathsf{CORR}, \mathsf{ISSUE}, \mathsf{OBTISS}, \mathsf{OBTAIN}, \mathsf{SIGN}$.

$\mathsf{Exp}^{\text{IND-CPA-}b}_{\text{E},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{E.Setup}(1^\kappa)$
$(ek, dk) \leftarrow \text{E.KG}(par)$
$b^* \leftarrow \mathcal{A}^{\mathsf{LR}(b,\cdot,\cdot)}(ek)$, where:
  $\mathsf{LR}(b, m_0, m_1)$ returns $\text{E.Enc}(ek, m_b)$
return b^*

$\mathsf{Exp}^{\text{IND-CCA-}b}_{\text{E},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{E.Setup}(1^\kappa)$
$(ek, dk) \leftarrow \text{E.KG}(par)$
$b^* \leftarrow \mathcal{A}^{\mathsf{LR}(b,\cdot,\cdot),\mathsf{DEC}(dk,\cdot)}(ek)$, where:
  $\mathsf{LR}(b, m_0, m_1)$ returns $\text{E.Enc}(ek, m_b)$
 and
  $\mathsf{DEC}(dk, c)$ returns $\text{E.Dec}(dk, c)$
    if c has not been output by LR
return b^*

Figure 14: IND-CPA and IND-CCA games.

B.2 Commitments

Although we don't directly leverage commitments functionality in our UAS scheme or constructions, they are an essential part of SBCM schemes, which we present in a following section. Thus, we overview commitment schemes briefly now. In a nutshell, a commitment scheme is defined by the following algorithms:

$par \leftarrow \text{C.Setup}(1^\kappa)$**.** Given a security parameter 1^κ, returns the public parameters par to commit messages.

$c \leftarrow \text{C.Commit}(par, m; r)$**.** Given the public parameters and a message m, outputs a commitment c to m, for which randomness r from some predefined randomness space $\mathcal{R}$ is used.

Opening a commitment c means revealing the message m and randomness r that were used to produce c. Commitment schemes are required to be binding and (usually) hiding:

Binding. Intuitively, the binding property of commitment schemes means that no adversary can change the message that has been committed to. More formally, $\Pr[\mathsf{Exp}^{\text{bind}}_{\text{C},\mathcal{A}}(1^\kappa) = 1]$ must be a negligible function of the security parameter.

Hiding. The hiding property captures that no adversary should be able to learn the message that was committed, when given only the commitment. This is formally defined through $\mathsf{Exp}^{\text{hide-}b}_{\text{C},\mathcal{A}}$, where $|\Pr[\mathsf{Exp}^{\text{hide-}b}_{\text{C},\mathcal{A}}(1^\kappa) = 1|b = 1] - \Pr[\mathsf{Exp}^{\text{hide-}b}_{\text{C},\mathcal{A}}(1^\kappa) = 1|b = 0]|$ must be a negligible function of the security parameter.

$\mathsf{Exp}^{\text{bind}}_{\text{C},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{C.Setup}(1^\kappa)$
$(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(par)$
$c_0 \leftarrow \text{C.Commit}(par, m_0, r_0)$
$c_1 \leftarrow \text{C.Commit}(par, m_1, r_1)$
if $m_0 \neq m_1 \wedge c_0 = c_1$: **return** 1
return 0

$\mathsf{Exp}^{\text{hide-}b}_{\text{C},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{C.Setup}(1^\kappa)$
$b' \leftarrow \mathcal{A}^{\mathsf{COM}(par,\cdot,\cdot)}$, where :
  $\mathsf{COM}(par, m_0, m_1)$:
    $r \xleftarrow{\$} \mathcal{R}$
    return $\text{C.Commit}(par, m_b, r)$
return b'

Figure 15: Games for commitment schemes.

Commitments on Blocks of Messages. We also use an extension of commitment schemes that allows committing to multiple messages at once. The properties we need are the same, and their definitions are extended in the natural way. Namely, C.Commit receives a vector/block of messages, $\boldsymbol{msg}$ instead of a single message. In the games, the adversary returns lets of messages and, in the binding game, the comparison $m_0 \neq m_1$ now compares lists $\boldsymbol{msg}_0$ and $\boldsymbol{msg}_1$, which must differ in at least one element. This extension is straightforward, for instance, from Pedersen commitments [BCC+15].

B.3 Simulation Extractable Non-Interactive Zero-Knowledge Proofs of Knowledge

Let $\mathcal{R}$ be an NP relation defined by pairs of elements (x, w), where x is a statement and w a witness proving that $(x, w) \in \mathcal{R}$. For concrete relations, we write $\mathcal{R} = \{(x), (w) : f(x, w)\}$, where $f(x, w)$ is a Boolean predicate denoting the concrete conditions that x and w need to meet. The set of all x such that there exists a w for which $(x, w) \in \mathcal{R}$ is the language, or $\mathcal{L}$, for $\mathcal{R}$. $x \notin \mathcal{L}$ means that there is no w such that $(x, w) \in \mathcal{R}$.

We use non-interactive zero-knowledge proofs of knowledge (NIZKPoK, or, for short, NIZK) over NP relations, in the Common Reference String (CRS) model. A NIZK system is a tuple (NIZK.Setup, NIZK.Prove, NIZK.Verify), defined as follows [GOS06]:

$crs \leftarrow \text{NIZK.Setup}(1^\kappa)$**.** Generates a CRS crs from security parameters 1^κ.

$\pi \leftarrow \text{NIZK.Prove}(crs, x, w)$**.** Given crs, statement x, and witness w, creates a proof π.

$1/0 \leftarrow \text{NIZK.Verify}(crs, \pi, x)$**.** Checks whether π is a valid proof for x.

Any zero-knowledge proof of knowledge must meet completeness, soundness, and zero-knowledge properties. We further need *simulation extractability* [CL06], and *extraction zero-knowledge* [GO14]. To define more formally the properties we need, we have to define three extra algorithms:

$(crs, \tau) \leftarrow \text{NIZK.}\mathsf{SimSetup}(1^\kappa)$**.** Produces a *crs* as the NIZK.Setup algorithm, along with a trapdoor τ.

$\pi \leftarrow \text{NIZK.}\mathsf{Sim}(crs, \tau, x)$**.** Given a trapdoor τproduced by NIZK.SimSetup, and a statement x, produces a simulated proof π.

$w = \text{NIZK.}\mathsf{Extract}(crs, \tau, x, \pi)$**.** Given a trapdoor τproduced by NIZK.SimSetup, and a proof π, returns a witness w. We assume (without loss of generality if one-way functions exist) that NIZK.Extract is deterministic.

When we want to make explicit the NP relation $\mathcal{R}$ to which the previous algorithms refer to, we use $\text{NIZK.Setup}^{\mathcal{R}}, \text{NIZK.Prove}^{\mathcal{R}}$, $\text{NIZK.Verify}^{\mathcal{R}}$, etc., and omit the NIZK prefix and super-index when clear from context. Altogether, the tuple $(\text{Setup}, \text{Prove}, \text{Verify}, \mathsf{SimSetup}, \mathsf{Sim}, \mathsf{Extract})$ needs to meet the following properties:

Completeness. Ensures that, for any $(x, w) \in \mathcal{R}$, any honest prover will be able to create a proof π that is accepted by any honest verifier, with overwhelming probability. More precisely, $\Pr[\mathsf{Exp}^{\text{comp}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1] = 0$, for any p.p.t. $\mathcal{A}$, for $\mathsf{Exp}^{\text{comp}}_{\text{NIZK},\mathcal{A}}$ in Fig. 16.

Soundness. Ensures that no adversary can create proofs accepted by Verify, for statements $x \notin \mathcal{L}$, except with negligible probability. That is, for for all p.p.t. $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\text{sound}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1]$ is negligible in 1^κ (where $\mathsf{Exp}^{\text{sound}}_{\text{NIZK},\mathcal{A}}$ is as in Fig. 16).

Zero-knowledge. Intuitively, captures that no information can be learned from a statement and proof pair, beyond the statement's validity. This is captured by requiring the adversary to distinguish between a run in the real world ($b = 0$), where the setup is done with Setup, and $\mathcal{A}$ has access to an honest prover Prove; and a run in an ideal world ($b = 1$), where the setup is replaced by SimSetup, and proofs are simulated with the help of the trapdoor produced by SimSetup. Note that, in the context of simulation extractable NIZK, this property not only requires that the simulated proofs are indistinguishable to the real ones; it also requires that SimSetup is indistinguishable from Setup. All this is formalized by requiring that $|\Pr[\mathsf{Exp}^{\text{zk-0}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{zk-1}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1]|$ be a negligible function of 1^κ, where $\mathsf{Exp}^{\text{zk-}b}_{\text{NIZK},\mathcal{A}}$ is as defined in Fig. 16.

Simulation Extractability. Simulation extractability is a stronger property than just soundness, combining knowledge soundness and simulation soundness. While soundness merely requires that the adversary cannot compute proofs for false statements, simulation extractability requires that an adversary cannot compute proofs for which it does not *know* a witness, even after seeing several simulated proofs. In our case, simulation extractability is straight-line (i.e. does not require rewinding), and adaptive (i.e. the adversary gets to see multiple extracted witnesses during its execution). Formally, for simulation extractability we require that $\Pr[\mathsf{Exp}^{\text{simext}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1]$ is a negligible function of 1^κ, where $\mathsf{Exp}^{\text{simext}}_{\text{NIZK},\mathcal{A}}$ is defined in Fig. 16.

As studied in [CL06], simulation extractable NIZKPoKs formalize the concept of "signatures of knowledge" (see, e.g., [CS97]). Which basically means that, given an (x, w) pair from an NP relation, we can treat x as a public key, and w as its corresponding private key, and leverage them to build digital signature schemes – with the advantage of being able to do so while proving arbitrary claims, as long as they can be represented as an NP relation. We note that, given a simulation extractable NIZK system, it is straightforward to build a signature of knowledge by adding the message to be signed in the statement of the NIZK.

Extraction zero-knowledge. While simulation-extractability models that "extraction still works in the presence of simulation", we will also require that "simulation still works in the presence of extraction". This is due to the fact that the UAS unforgeability games are built on extraction to decide the winning condition, and then in the security proof (e.g., Theorem 3), we need to argue that we can still apply zero-knowledge to the game that already uses the extractor. This is a non-standard property, but has been mentioned before [GO14]. We will also sketch how to implement it. More formally, we give an adversary access to an oracle that outputs either honest or simulated proofs (like for the zero-knowledge property), *as well as* to an extraction oracle, with the restriction that the adversary must not ask for extraction of proofs returned by the first oracle (since that would enable a trivial distinguisher given that we can extract a witness from an honest proof but cannot generally extract from a simulated proof). Formally, for all adversaries $\mathcal{A}$, we require that $|\Pr[\mathsf{Exp}^{\text{extzk-0}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{extzk-1}}_{\text{NIZK},\mathcal{A}}(1^\kappa) = 1]|$ be a negligible function of 1^κ, where $\mathsf{Exp}^{\text{extzk-}b}_{\text{NIZK},\mathcal{A}}$ is as defined in Fig. 16. A similar (in spirit) definition has been given in [GO14].

$\mathsf{Exp}^{\text{comp}}_{\text{NIZK},\mathcal{A}}(1^\kappa)$

$crs \leftarrow \text{Setup}(1^\kappa)$
$(x, w) \leftarrow \mathcal{A}(crs)$
if $(x, w) \notin \mathcal{R}$: **return** 0
$\pi \leftarrow \text{Prove}(crs, x, w)$
$b \leftarrow \text{Verify}(crs, \pi, x)$
return $1 - b$

$\mathsf{Exp}^{\text{simext}}_{\text{NIZK},\mathcal{A}}(1^\kappa)$

$(crs, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$
$\mathcal{A}^{\mathsf{Sim}(crs,\tau,\cdot),\mathsf{Extract}(crs,\tau,\cdot,\cdot)}(crs)$
return 1 if for some $w \leftarrow \mathsf{Extract}(crs, \tau, x, \pi)$
query, $\text{Verify}(crs, \pi, x) = 1 \wedge (x, w) \notin \mathcal{R} \wedge$
no $\mathsf{Sim}(\cdot, \cdot, x)$ query resulted in π

$\mathsf{Exp}^{\text{extzk-}b}_{\text{NIZK},\mathcal{A}}(1^\kappa)$

$(crs, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$
if $b = 0$:
return $\mathcal{A}^{\text{Prove}(crs,\cdot,\cdot),\mathsf{Extract}'(crs,\tau,\cdot,\cdot)}(crs)$
if $b = 1$:
return $\mathcal{A}^{\mathsf{Sim}'(crs,\tau,\cdot,\cdot),\mathsf{Extract}'(crs,\tau,\cdot,\cdot)}(crs)$

$\mathsf{Exp}^{\text{sound}}_{\text{NIZK},\mathcal{A}}(1^\kappa)$

$crs \leftarrow \text{Setup}(1^\kappa)$
$(x, \pi) \leftarrow \mathcal{A}(crs)$
return $x \notin \mathcal{L} \wedge \text{Verify}(crs, \pi, x) = 1$

$\mathsf{Exp}^{\text{zk-}b}_{\text{NIZK},\mathcal{A}}(1^\kappa)$

if $b = 0$:
$crs \leftarrow \text{Setup}(1^\kappa)$
return $\mathcal{A}^{\text{Prove}(crs,\cdot,\cdot)}(crs)$
if $b = 1$:
$(crs, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$
return $\mathcal{A}^{\mathsf{Sim}'(crs,\tau,\cdot,\cdot)}(crs)$

$\mathsf{Sim}'(crs, \tau, x, w)$

if (x, w) :
return $\mathsf{Sim}(crs, \tau, x)$
return $\perp$

$\mathsf{Extract}'(crs, \tau, x, \pi)$

if no $\text{Prove}(crs, x, \cdot)$ or $\mathsf{Sim}'(crs, x, \cdot)$ query resulted in π :
return $\text{NIZK.Extract}(crs, \tau, x, \pi)$
return $\perp$

Figure 16: Games for Simulation Extractable NIZK schemes.

Implementing a proof that is zero-knowledge in the presence of an extraction oracle. One can upgrade a traditional straight-line adaptively simulation extractable NIZK to one that is additionally zero-knowledge in the presence of an extraction oracle. The construction for this is effectively encrypt-then-prove.

In more detail, let $\mathcal{R}$ be some relation, let E be some public-key encryption scheme, and let NIZK$'$ be a non-interactive proof system for relation $\mathcal{R}' = \{(x' = (x, c, par_{\mathrm{E}}, ek), w' = (w, r)) \mid (x, w) \in \mathcal{R} \wedge c = \mathrm{E.Enc}(ek, w; r)\}$. We construct NIZK as follows.

$crs \leftarrow \mathrm{NIZK.Setup}(1^\kappa)$**.** Runs $crs' \leftarrow \mathrm{NIZK'.Setup}(1^\kappa)$ and $par_{\mathrm{E}} \leftarrow \mathrm{E.Setup}(1^\kappa)$, $(ek, dk) \leftarrow \mathrm{E.KG}(par_{\mathrm{E}})$. Sets $crs = (crs', par_{\mathrm{E}}, ek)$.

$\pi \leftarrow \mathrm{NIZK.Prove}(crs, x, w)$**.** Computes $c = \mathrm{E.Enc}(ek, w; r)$ for random r, then $\pi' \leftarrow \mathrm{NIZK'.Prove}(crs', (x, c, par_{\mathrm{E}}, ek), (w, r))$. It outputs $\pi = (\pi', c)$.

$1/0 \leftarrow \mathrm{NIZK.Verify}(crs, \pi, x)$**.** For $\pi = (\pi', c)$ checks that $\mathrm{NIZK.Verify}(crs', \pi', (x, c, par_{\mathrm{E}}, ek)) = 1$.

For this construction, we can use the following simulator/extractor helpers.

$(crs, \tau) \leftarrow \mathrm{NIZK.}\mathsf{SimSetup}(1^\kappa)$**.** Runs $(crs', \tau') \leftarrow \mathrm{NIZK'.}\mathsf{SimSetup}(1^\kappa)$ and $par_{\mathrm{E}} \leftarrow \mathrm{E.Setup}(1^\kappa)$, $(ek, dk) \leftarrow \mathrm{E.KG}(par_{\mathrm{E}})$. Sets $crs = (crs', par_{\mathrm{E}}, ek)$ and $\tau = \tau'$.

$\pi \leftarrow \mathrm{NIZK.}\mathsf{Sim}(crs, \tau, x)$**.** Computes $c = \mathrm{E.Enc}(ek, 0)$ (where 0 is some appropriate constant value), then $\pi' \leftarrow \mathrm{NIZK'.}\mathsf{Sim}(crs', \tau, (x, c, par_{\mathrm{E}}, ek))$. It outputs $\pi = (\pi', c)$.

$w = \mathrm{NIZK.}\mathsf{Extract}(crs, \tau, x, \pi)$**.** For $\pi = (\pi', c)$, runs $(w, r) = \mathrm{NIZK'.}\mathsf{Extract}(crs', \tau, (x, c, par_{\mathrm{E}}, ek), \pi')$ and outputs w.

It is easy to see that NIZK as described above inherits completeness (if the encryption scheme E is correct), zero-knowledge (if E is CPA-secure), and simulation extractability from NIZK$'$.

We briefly sketch how to prove zero-knowledge in the presence of an extraction oracle for NIZK. Assume that NIZK$'$ is zero-knowledge and simulation extractable and that the encryption scheme E is correct and CPA-secure. Our rough strategy will be to switch to using the ciphertext c for extraction so that we can apply ZK without having to extract from proofs. We start with $\mathsf{Exp}^{\text{extzk-0}}_{\mathrm{NIZK},\mathcal{A}}(1^\kappa)$ and game-hop our way to $\mathsf{Exp}^{\text{extzk-1}}_{\mathrm{NIZK},\mathcal{A}}(1^\kappa)$ as follows:

1. Instead of running NIZK$'$.$\mathsf{Extract}$ in the extraction oracle, decrypt c and output the plaintext. This change is undetectable because of the soundness of NIZK$'$ and the correctness of the encryption scheme.

2. Switch from computing π via NIZK$'$.Prove in the first oracle to computing π via NIZK$'$.Sim. This change is undetectable because of the zero-knowledge property of NIZK$'$.

3. Switch extraction back, i.e. instead of decrypting c during extraction queries, compute the witness via NIZK$'$.$\mathsf{Extract}$ again. This change is undetectable because of simulation extractability of NIZK$'$.

4. Switch the ciphertext of simulated proofs to encryptions of 0, i.e. instead of $c = \mathrm{E.Enc}(ek, w)$, compute $c = \mathrm{E.Enc}(ek, 0)$ during simulation queries. This change is undetectable because of CPA-security of E (note that due to the previous game-hop, the experiment does not need the decryption key for E anymore, hence CPA-security suffices).

The last change brings the modified game in line with $\mathsf{Exp}^{\text{extzk-1}}_{\text{NIZK},\mathcal{A}}(1^\kappa)$. In a way, this proof is reminiscent of Naor-Yung's proof of a CCA secure encryption scheme using NIZKs [NY90], which is natural given that zero-knowledge in the presence of an extraction oracle has a sort of "CCA" flavor.

B.4 Signatures over Blocks of Committed Messages

For our generic constructions, we use interactive signing protocols between a user and a signer, where the user has a block of messages to sign blindly, and both receive a common block of messages to be also included in the resulting signature. This is precisely the case of partially blind signatures, that collapse to blind signatures [PS96] when there is no common message between user and signer; and to conventional signatures when the user does not input a message to be blindly signed [AO00].

To the best of our knowledge, models of existing schemes for signing blocks of messages like [CL02, ASM06, PS16, CDL16] target the case of signing blocks of *plain* messages, and are subsequently informally extended to support signing commitments to blocks of messages via interactive protocols. However, they do not support signing both committed and plain messages (although the extension is trivial); and, more importantly, do not give security models of the resulting construction, nor of course prove its security. As we use this variant as a generic building block, we briefly model such a scheme for Signatures over Blocks of Committed Messages (SBCM).

The syntax for an SBCM scheme is as follows:

$par \leftarrow \text{SBCM.Setup}(1^\kappa)$**.** It produces public parameters for the other algorithms, given an input security parameter 1^κ.

$(vk, sk) \leftarrow \text{SBCM.KG}(par)$**.** Generates a verification-signing key pair.

$c \leftarrow \text{SBCM.Blind}(vk, \overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$**.** A user computes commitment c to request a signature over messages $\overline{\boldsymbol{msg}}$ (in committed form) and $\boldsymbol{msg}$ (in plain form), to signer with verification key vk. r is expected to be a random value $r \leftarrow \Gamma$ from a predefined randomness set Γ. The output is the commitment c.

$\beta \leftarrow \text{SBCM.Sign}(sk, c, \boldsymbol{msg})$**.** The signer, with signing key sk, produces a partial signature β over the messages committed to in commitment c, as well as the messages in $\boldsymbol{msg}$.

$\sigma \leftarrow \text{SBCM.Unblind}(vk, \beta, c, r, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$**.** A user who requested a signature over $\overline{\boldsymbol{msg}}$ and $\boldsymbol{msg}$, where c is a commitment over $\overline{\boldsymbol{msg}}$ using randomness r, finalizes the signature, computing σ from the signer's partial signature β.

$1/0 \leftarrow \text{SBCM.Verify}(vk, \sigma, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$**.** Checks whether σ is a valid signature over the set of messages $\overline{\boldsymbol{msg}}$ and $\boldsymbol{msg}$, under verification key vk.

The correctness and security properties are defined as follows.

Correctness. Informally, an SBCM scheme is correct if signatures generated between an honest party running SBCM.Blind, an honest signer running SBCM.Sign fed with the output of SBCM.Blind and matching $\boldsymbol{msg}$ and signing key pair, and the user finally running SBCM.Unblind over the partial signature by the signer and leveraging the same randomness as in SBCM.Blind, produces a signature over $\overline{\boldsymbol{msg}}$ and $\boldsymbol{msg}$ that is accepted by SBCM.Verify.

Deterministically derived public keys. Because SBCM keys are not only used to issue credentials, but also serve as user keys $(vk, sk) = (upk, usk)$, we require that a user's secret key usk has a unique public key upk associated with it. More formally, we require that there is a deterministic function f such that for all $(vk, sk) \in [\text{SBCM.KG}(par)]$, we have $vk = f(par, sk)$. In theory, this is not a restriction (the SBCM.KG randomness can serve as a canonical sk). In practice, most SBCM schemes are already of that form where SBCM.KG first generates random values for the secret key and then deterministically computes the corresponding public key.

Unforgeability. It must be unfeasible for an adversary to produce signatures over blocks of messages that have not been signed (in committed shape) by the signer. In order to enable us to even decide which messages have been (blindly) signed, we force the adversary to reveal messages $\overline{\boldsymbol{msg}}, \boldsymbol{msg}$ and commitment randomness r whenever it wants to query a signature. This gives us security against adversaries $\mathcal{A}$ that *know* what messages they request, which, when using SBCM as a building block in a larger construction, can be achieved by making $\mathcal{A}$ prove knowledge of the the messages. More formally, an SBCM scheme is unforgeable if, for all p.p.t. adversaries $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\text{EUF}}_{\text{SBCM},\mathcal{A}}(1^\kappa) = 1]$, as defined in Fig. B.4, is a negligible function of the security parameter.

Blindness. Finally, the signer must not learn the plaintext values of the messages that are signed in committed form. Note that this is a weaker notion than the usual blindness property of (partially) blind signature schemes, where it is additionally required that the adversary cannot link a signature to the signing process that produced it. Informally, we capture this basically as the hiding notion of a commitment scheme – and formally define it in $\mathsf{Exp}^{\text{blind-b}}_{\text{SBCM},\mathcal{A}}$ in Fig. B.4. Note that, in the definition, we explicitly do not give back to the adversary $\mathcal{A}$ any full signature (i.e., after running SBCM.Unblind) obtained from values returned by $\mathcal{A}$, as this would allow the adversary to trivially check what messages (among the ones he chose) were signed. While this may seem a too weak notion, it is good enough for our needs, as in our UAS construction we never share actual signatures, but zero-knowledge proofs of knowledge of such signatures.

An SBCM scheme is blind if, for all p.p.t. adversaries $\mathcal{A}$, $|\Pr[\mathsf{Exp}^{\text{blind-1}}_{\text{SBCM},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{blind-0}}_{\text{SBCM},\mathcal{A}}(1^\kappa) = 1]|$ is a negligible function of the security parameter.

$\mathsf{Exp}^{\text{EUF}}_{\text{SBCM},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{Setup}(1^\kappa)$
$(vk, sk) \leftarrow \text{KG}(par)$
$(\sigma^*, \overline{\boldsymbol{msg}}^*, \boldsymbol{msg}^*) \leftarrow \mathcal{A}^{\mathsf{SIGN}(\cdot,\cdot,\cdot)}(par, vk)$
 where $\mathsf{SIGN}(\overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$:
  $c \leftarrow \text{SBCM.Blind}(vk, \overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$
  $\beta \leftarrow \text{SBCM.Sign}(sk, c, \boldsymbol{msg})$
  return β
if $\text{SBCM.Verify}(vk, \sigma^*, \overline{\boldsymbol{msg}}^*, \boldsymbol{msg}^*) = 1$
  and $\mathcal{A}$ did not query $\mathsf{SIGN}(\overline{\boldsymbol{msg}}^*, \boldsymbol{msg}^*, \cdot)$:
  return 1
return 0

$\mathsf{Exp}^{\text{blind-b}}_{\text{SBCM},\mathcal{A}}(1^\kappa)$

$par \leftarrow \text{Setup}(1^\kappa)$
$(vk, sk) \leftarrow \text{KG}(par)$
$b^* \leftarrow \mathcal{A}^{\mathsf{BLIND}(\cdot,\cdot,\cdot)}(par, vk)$, where:
  $\mathsf{BLIND}(\overline{\boldsymbol{msg}}_0, \overline{\boldsymbol{msg}}_1, \boldsymbol{msg})$:
   $r \leftarrow \mathcal{R}$
   $c \leftarrow \text{Blind}(vk, \overline{\boldsymbol{msg}}_b, \boldsymbol{msg}, r)$;
   Hand c to $\mathcal{A}$, receive β from $\mathcal{A}$
   $\sigma \leftarrow \text{SBCM.Unblind}(vk, \beta, c, r, \overline{\boldsymbol{msg}}_b, \boldsymbol{msg})$
   return $\text{SBCM.Verify}(vk, \sigma, \overline{\boldsymbol{msg}}_b, \boldsymbol{msg})$
return b^*

Figure 17: Games for SBCM schemes.

B.4.1 An Instantiation of SBCM with BBS+

Next, we give an instantiation of an SBCM scheme, based on BBS+ signatures. We emphasize again that this is essentially equivalent to the protocol for signing committed block of messages in [ASM06] and, also, to the equivalent ones in [CL02, PS16] (although not for BBS+ signatures). The main difference being that we allow merging committed blocks of messages and blocks of (plaintext) messages into the same signature.

$par \leftarrow$ SBCM.Setup$(1^\kappa, n, \overline{n})$**.** Generates a bilinear group $\mathbb{B} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{G}(1^\kappa)$, and $n + \overline{n} + 1$ additional generators g, $h_1, ..., h_n, \overline{h}_1, ..., \overline{h}_{\overline{n}}$ of $\mathbb{G}_1$. Returns $par \leftarrow (1^\kappa, n, \overline{n}, \mathbb{B}, g, h_1, ..., h_n, \overline{h}_1, ..., \overline{h}_{\overline{n}})$. We assume that par is available to all other algorithms, even when not explicitly passed as an argument.

$(vk, sk) \leftarrow$ SBCM.KG(par)**.** Parses par as $(1^\kappa, \cdot, \cdot, (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e), \dots)$. Outputs $sk \leftarrow \mathbb{Z}_p^*$, and $vk = g_2^{sk}$.

$c \leftarrow$ SBCM.Blind$(vk, \overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$**.** If $|\boldsymbol{msg}| > n$ or $|\overline{\boldsymbol{msg}}| > \overline{n}$, abort. Else, compute $c \leftarrow g^r \prod_{i \in [|\overline{\boldsymbol{msg}}|]} \overline{h}_i^{\overline{\boldsymbol{msg}}_i}$. Output c.

$\beta \leftarrow$ SBCM.Sign$(sk, c, \boldsymbol{msg})$**.** If $|\boldsymbol{msg}| > n$, abort. Else, compute $x, \tilde{s} \xleftarrow{\$} \mathbb{Z}_p^*, A \leftarrow (g_1 c g^{\tilde{s}} \prod_{i \in |\boldsymbol{msg}|} h_i^{\boldsymbol{msg}_i})^{1/(sk+x)}$. Return $\beta \leftarrow (A, x, \tilde{s})$.

$\sigma \leftarrow$ SBCM.Unblind$(vk, \beta, c, r, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$**.** Parse β as $(A, x, \tilde{s})$ If $A = 1_{\mathbb{G}_1}$: return $\perp$. If $e(A, vk \cdot g_2^x) \neq e(g_1 c g^{\tilde{s}} \prod_{i \in |\boldsymbol{msg}|} h_i^{\boldsymbol{msg}_i}, g_2)$: return $\perp$. Else, set $s \leftarrow r + \tilde{s}$ and return (A, x, s).

$1/0 \leftarrow$ SBCM.Verify$(vk, \sigma, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$**.** To verify a signature σ, for message set $\overline{\boldsymbol{msg}}$ that was signed as a block commitment, and message set $\boldsymbol{msg}$, signed as plaintext, parse σ as (A, x, s) and check that

$$e(A, g_2^x vk) = e\left(g_1 g^s \cdot \prod_{i \in |\overline{\boldsymbol{msg}}|} \overline{h}^{\overline{\boldsymbol{msg}}_i} \cdot \prod_{i \in |\boldsymbol{msg}|} h^{\boldsymbol{msg}_i}, g_2\right)$$

Proving that a key pair was correctly generated. In $\mathcal{R}_{\text{is}}$ and $\mathcal{R}_{\text{ev}}$ of Π_{UAS}, we include statements requiring to prove that a key pair was produced by a KG algorithm (therein denoted $(upk, usk) \in [\text{KG}(par)]$). In the case of the BBS+ construction of SBCM, this is as simple as proving knowledge of a discrete logarithm.

Proving Knowledge of Signature. Proving knowledge of a BBS+ signature as produced in our SBCM variant is essentially the same as in [ASM06, CDL16], only needing to account for the different basis for messages signed in committed and plain form.

Correctness. Correctness is easy to verify.

EUF security. EUF security of our SBCM variant is easily derivable from EUF security of vanilla BBS+ signatures [CDL16]. We sketch an adversary $\mathcal{A}$ against BBS+, given an adversary $\mathcal{B}$ against SBCM. To help the explanation, we point out that a vanilla BBS+ scheme is just like SBCM, but removing the Blind and Unblind algorithms, and BBS.Sign takes the full message in plain form (instead of partially committed). Given the previous, $\mathcal{A}$ operates as follows. It first receives the BBS+ challenge: an (par, vk) pair, where $(1^\kappa, n, \mathbb{B}, g, h_1, ..., h_n) \leftarrow par$ for some bilinear group $\mathbb{B}$, and vk has the exact same structure as in SBCM. To simulate the environment for $\mathcal{B}$, $\mathcal{A}$ appropriately divides n in $n_1 \geq 1$ and $n_2 \geq 1$ (such that $n_1 + n_2 = n$), and separates the generator set consequently, to produce a parameter set par' with the structure expected by $\mathcal{B}$. That is, this produces $par' \leftarrow (1^\kappa, n_1, n_2, \mathbb{B}, g, h_1, ..., h_{n_1}, \overline{h}_1, ..., \overline{h}_{\overline{n}_2})$. It is direct that par' is indistinguishable from the output of SBCM.Setup. Then, $\mathcal{A}$ invokes $\mathcal{B}$ on (par', vk). To simulate $\mathcal{B}$'s queries to its $\mathsf{SIGN}(\overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$ oracle, $\mathcal{A}$ queries its oracle $\mathrm{Sign}(\overline{\boldsymbol{msg}}, \boldsymbol{msg})$ to receive (A, x, s), where $x, s \xleftarrow{\$} \mathbb{Z}_p^*$ and $A = (g_1 \cdot g^s \cdot \prod_{i \in |\overline{\boldsymbol{msg}}|} \overline{h}^{\overline{\boldsymbol{msg}}_i} \cdot \prod_{i \in |\boldsymbol{msg}|} h^{\boldsymbol{msg}_i})^{1/(sk+x)}$, and then $\mathcal{A}$ returns the appropriately "blinded" signature $\beta = (A, x, s - r)$ to $\mathcal{B}$. Note that β is distributed as a legitimate partial signature produced by a SIGN oracle. $\mathcal{A}$ outputs whatever $\mathcal{B}$ does. It is easy to see that, whenever $\mathcal{B}$ wins (against SBCM EUF), then so does $\mathcal{A}$ (against BBS+ EUF).

Blindness. Blindness is a direct consequence of the hiding property of the underlying Pedersen block commitment scheme. Indeed, the distribution of $c = g^r \prod_{i \in [|\overline{\boldsymbol{msg}}|]} \overline{h}_i^{\overline{\boldsymbol{msg}}_i}$ as output by $\mathrm{SBCM.Blind}(vk, \overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$ is uniformly random, independent of $\overline{\boldsymbol{msg}}$, given that r is uniformly random in $\mathbb{Z}_p$. Furthermore, the issuer does not learn anything about the messages from the signature verification bit returned by the BLIND oracle. This is because that bit can be predicted by the issuer, using only public information. It will be 1 if and only the issuer's $\beta = (A, x, \tilde{s})$ fulfills $e(A, vk \cdot g_2^x) = e(g_1 c g^{\tilde{s}} \prod_{i \in |\boldsymbol{msg}|} h_i^{\boldsymbol{msg}_i}, g_2)$ (note that this expression does not involve the hidden messages $\overline{\boldsymbol{msg}}$, only the public commitment c and public messages $\boldsymbol{msg}_i$). As a result, blindness holds perfectly.

C Correctness and Security Proofs for Π_{UAS}

To prove security, we use the following helpers (as required by the theorems):

$\mathsf{SimSetup}(1^\kappa) \rightarrow (par, \tau)$**.** Runs $par_{\mathrm{SBCM}} \leftarrow \mathrm{SBCM.Setup}(\kappa)$, $par_{\mathrm{E}} \leftarrow \mathrm{E.Setup}(\kappa)$, $(crs_{\mathrm{is}}, \tau_{\mathrm{is}}) \leftarrow \mathrm{NIZK.SimSetup}^{\mathcal{R}_{\mathrm{is}}}(\kappa)$, $(crs_{\mathrm{ev}}, \tau_{\mathrm{ev}}) \leftarrow \mathrm{NIZK.SimSetup}^{\mathcal{R}_{\mathrm{ev}}}(\kappa)$, and $(crs_{\mathrm{op}}, \tau_{\mathrm{op}}) \leftarrow \mathrm{NIZK.SimSetup}^{\mathcal{R}_{\mathrm{op}}}(\kappa)$. Return $par = (par_{\mathrm{SBCM}}, par_{\mathrm{E}}, crs_{\mathrm{is}}, crs_{\mathrm{ev}}, crs_{\mathrm{op}})$ and $\tau = (\tau_{\mathrm{is}}, \tau_{\mathrm{ev}}, \tau_{\mathrm{op}})$.

$\mathsf{ExtIss}(\tau, reg) \rightarrow (upk, \{(cid_i, \boldsymbol{a}_i)\}_{i \in [n]})$**.** Parses $reg = (cid, c, \pi)$. Runs $(upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i \in [n]}, r) \leftarrow \mathrm{NIZK.Extract}^{\mathcal{R}_{\mathrm{is}}}(crs_{\mathrm{is}}, \tau_{\mathrm{is}}, (f_{\mathrm{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i \in [n]}, y_{\mathrm{is}}), \pi)$. Outputs $(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i \in [n]})$.

$\mathsf{ExtSign}(\tau, \Sigma) \rightarrow (upk, \{(cid_i, \boldsymbol{a}_i)\}_{i \in [n]})$**.** Parses $\Sigma = ((\pi_{\mathrm{ev}}, c_{\mathrm{op}}), y_{\mathrm{ev}})$. Runs $(upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i \in [n]}, y_{\mathrm{op}}, r) \leftarrow \mathrm{NIZK.Extract}^{\mathcal{R}_{\mathrm{ev}}}(crs_{\mathrm{ev}}, \tau_{\mathrm{ev}}, (m, f_{\mathrm{ev}}, y_{\mathrm{ev}}, f_{\mathrm{op}}, c_{\mathrm{op}}, \{ipk_i\}_{i \in [n]}, ek), \pi_{\mathrm{ev}})$. Outputs $(upk, \{(cid_i, \boldsymbol{a}_i)\}_{i \in [n]})$.

Theorem 6 (Correctness of Π_{UAS})**.** *If the underlying schemes for public-key encryption and* SBCM *are correct, and the* NIZK *is complete, then our generic construction* Π_{UAS} *satisfies correctness as defined in Definition 1.*

Correctness of Π_{UAS} follows easily from inspection, given that the involved signature and opening proof are honestly computed. However, the adversary can also win the correctness game if it finds a collision in the credential identifiers (i.e., $coll = 1$). Hence, it is worth to emphasize that $coll = 1$ only with negligible probability. To see this, recall that $coll = 1$ if there is a collision in the cid of one or more credentials created during the $\langle \mathrm{Obt}, \mathrm{Iss} \rangle$ protocol – this can be checked in the issuance-related oracles, which contain checks such as "**if** $cid \in \mathsf{CRD}$: $coll \leftarrow 1$". Note that, in Π_{UAS}'s $\langle \mathrm{Obt}, \mathrm{Iss} \rangle$ protocol, $cid = (cid^{\mathrm{I}}, cid^{\mathrm{U}})$, where the issuer picks cid^{I}, and the user picks cid^{U}. Since, in the oracles that set $coll$, at least one of them is always picked uniformly at random from the attribute space $\mathcal{AS}$ (which is assumed to be of length dependent on the security parameter), then the probability of finding a collision is negligible in the security parameter.

Theorem 1 (Issuance anonymity of Π_{UAS})**.** *If the* SBCM *scheme is blinding, the NIZK system is zero-knowledge and simulation-extractable, and the public-key encryption scheme is correct and IND-CPA secure, then* Π_{UAS} *satisfies issuance anonymity as defined in Definition 2.*

For the proof, intuitively our goal is to show that we can assign all challenge credentials to a virtual user (upk^*, usk^*), independent of the bit b. To enable the switch, we need to simulate the NIZK proofs (among others, to ensure that the adversary cannot notice the switch from looking at proofs). Another way the adversary may learn about b is by breaking the encryption of y_{op} when SIGNing with challenge credentials, so we replace the encryption c_{op} with an encryption of 0 and argue that the adversary cannot notice this change because of IND-CPA security. To enable the latter argument, we need to change the OPEN oracle to reply using the NIZK extractor instead of the decryption key to compute its answers.

Theorem 1. Consider the following sequence of games.

$\mathsf{G}_0^b = \mathsf{Exp}_{\mathsf{UAS},\mathcal{A}}^{\text{iss-anon-}b}(1^\kappa)$ is the original game from Definition 2.

G_1^b works like G_0, except that

- During SIGN queries, whenever a value is added to SIG, the experiment additionally remembers $(\mathsf{oid}, \Sigma, y_{\mathrm{op}})$, where oid is the opener id passed as SIGN input, Σ is the resulting overall UAS signature returned by the oracle, and $y_{\mathrm{op}} = f_{\mathrm{op}}(\mathsf{UPK}[\mathsf{uid}], (\boldsymbol{cid}, \mathsf{ATT}[\boldsymbol{cid}], m))$ is the opening value computed by the Sign algorithm.
- When $\mathsf{OPEN}(\mathsf{oid}, \mathbf{iid}, \Sigma, m, f_{\mathrm{ev}})$ is queried such that $(\cdot, \mathsf{oid}, \cdot, \cdot, \Sigma) \in \mathsf{SIG}$, then instead of computing $y_{\mathrm{op}} = \mathrm{E.Dec}(osk, c_{\mathrm{op}})$, we set y_{op} to the value remembered for (oid, Σ) during its corresponding SIGN query.

From the point of view of $\mathcal{A}$, there is no difference between G_0^b and G_1^b given the correctness property of the encryption scheme. Hence $\Pr[\mathsf{G}_0^b = 1] = \Pr[\mathsf{G}_1^b = 1]$.

G_2^b works like G_1^b, except that:

- $par \leftarrow$ Setup is replaced with $(par, \tau) \leftarrow \mathsf{SimSetup}$ (where $\mathsf{SimSetup}$ is defined as above).
- Invocations of $\pi \leftarrow \mathrm{NIZK.Prove}^{\mathcal{R}_{\mathrm{is}}}(crs_{\mathrm{is}}, (f_{\mathrm{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\mathrm{is}}), \cdot)$ are replaced with $\pi \leftarrow \mathrm{NIZK.Sim}^{\mathcal{R}_{\mathrm{is}}}(crs_{\mathrm{is}}, \tau_{\mathrm{is}}, (f_{\mathrm{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\mathrm{is}}))$ (this happens in the $\mathsf{OBTISS}, \mathsf{OBTAIN}, \mathsf{OBTCHAL}_b$ oracles).

- Invocations of $\pi_{\text{ev}} \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), \cdot)$ are replaced with $\pi_{\text{ev}} \leftarrow \text{NIZK.}\mathsf{Sim}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, \tau_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek))$ (this happens in the SIGN oracle).
- Invocations of $\pi_{\text{op}} \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{op}}}(crs_{\text{op}}, (opk, c, y_{\text{op}}), (osk))$ are replaced with $\pi_{\text{op}} \leftarrow \text{NIZK.}\mathsf{Sim}^{\mathcal{R}_{\text{op}}}(crs_{\text{op}}, \tau_{\text{op}}, (opk, c, y_{\text{op}}))$ (this happens in the OPEN oracle).

Through three straightforward reductions $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ (in a hybrid fashion, incrementally replacing the proofs/setup for each of the three relations), one can show that $|\Pr[\mathsf{G}_1^b = 1] - \Pr[\mathsf{G}_2^b = 1]| \leq \sum_{i=1}^{3} |\Pr[\mathsf{Exp}^{\text{zk-0}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{zk-1}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1]|$ is negligible. For that reduction, note that by design of the games, whenever we invoke NIZK.Prove, the witness used is valid.

G_3^b works like G_2^b, except that during OPEN queries, whenever we would compute $y_{\text{op}} = \text{E.Dec}(osk, c_{\text{op}})$, G_3^b instead computes y_{op} as $(\cdot, \cdot, \cdot, y_{\text{op}}, r) = \text{NIZK.}\mathsf{Extract}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, \tau_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), \pi_{\text{ev}})$.
Let $\mathsf{PoKfail}$ be the event that $\text{NIZK.}\mathsf{Extract}^{\mathcal{R}_{\text{ev}}}$ at some point outputs an invalid witness. Note that if $\mathsf{PoKfail}$ does *not* occur, then there is no difference between G_3^b and G_2^b (since the decryption result is the same as the extracted y_{op}, as guaranteed by correctness of encryption and the relation $\mathcal{R}_{\text{ev}}$). Through a straightforward reduction $\mathcal{B}$, one can show that $|\Pr[\mathsf{G}_2^b = 1] - \Pr[\mathsf{G}_3^b = 1]| \leq \Pr[\mathsf{PoKfail}] = \Pr[\mathsf{Exp}^{\text{simext}}_{\text{NIZK},\mathcal{B}}(\kappa) = 1]$ is negligible. For the reduction, note that by definition of the OPEN oracle and the change in G_1^b, we only apply extraction if $(\cdot, \mathsf{oid}, \cdot, \cdot, \Sigma) \notin \mathsf{CSIG} \cup \mathsf{SIG}$. Hence we never try to extract from simulated signatures.

G_4^b works like G_3^b, except that during SIGN queries, if $\mathsf{oid} \notin \mathsf{CO}$ and $cid_1 \in \mathsf{CCRD}$, then instead of $c_{\text{op}} \leftarrow \text{E.Enc}(ek, y_{\text{op}}; r)$, the oracle computes $c_{\text{op}} \leftarrow \text{E.Enc}(ek, 0; r)$ (for some fixed message "0" in the encryption scheme's message space).
Through a straightforward reduction $\mathcal{B}$ that replaces the c_{op} ciphertexts in SIGN for the first $i \leftarrow \{1, \ldots, p(\kappa)\}$ openers in a hybrid fashion, one can show that $|\Pr[\mathsf{G}_3^b = 1] - \Pr[\mathsf{G}_4^b = 1]| \leq p(\kappa) \cdot |\Pr[\mathsf{Exp}^{\text{IND-CPA-0}}_{\text{E},\mathcal{B}}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{IND-CPA-1}}_{\text{E},\mathcal{B}}(\kappa) = 1]|$, where p is a polynomial upper-bound for the number of honest openers created by $\mathcal{A}$, is negligible. For the reduction, note that because of the changes in G_1 and G_3, the decryption key not used during OPEN queries anymore. The decryption key has to be exposed when the opener is corrupted via OCORR, but whenever our modification takes place, then an entry containing oid is added to CSIG and hence $\mathsf{OCORR}(\mathsf{oid})$ just returns $\perp$.

G_5^b works like G_4^b except that:

- In the beginning, it generates an additional virtual user key pair $(upk^{\perp}, usk^{\perp}) \leftarrow \text{KG}(par)$.
- During $\mathsf{OBTCHAL}_b$ queries, invocations of $c = \text{SBCM.Blind}(ipk, usk, (cid, \boldsymbol{a}), r)$ are replaced with $c = \text{SBCM.Blind}(ipk, \underline{usk^{\perp}}, (cid, \boldsymbol{a}), r)$.
- During $\mathsf{OBTCHAL}_b$ queries, invocations of $\sigma \leftarrow \text{SBCM.Unblind}(ipk, \beta, c, r, usk^*, (cid, \boldsymbol{a}))$ are replaced with $\sigma \leftarrow \text{SBCM.Unblind}(ipk, \beta, c, r, \underline{usk^{\perp}}, (cid, \boldsymbol{a}))$.
- During $\mathsf{OBTCHAL}_b$ queries, invocations of $\text{SBCM.Verify}(ipk, \sigma, usk, (cid, \boldsymbol{a}))$ are replaced with $\text{SBCM.Verify}(ipk, \sigma, \underline{usk^{\perp}}, (cid, \boldsymbol{a}))$.

Through a straightforward reduction $\mathcal{B}$, one can show that $|\Pr[\mathsf{G}_4^b = 1] - \Pr[\mathsf{G}_5^b = 1]| = |\Pr[\mathsf{Exp}^{\text{blind-0}}_{\text{SBCM},\mathcal{B}}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{blind-1}}_{\text{SBCM},\mathcal{B}}(\kappa) = 1]|$ is negligible. Note that in G_5^b, challenge

credential CCRD signatures *crd* are not valid (they contain the wrong user secret $usk^{\perp}$), but they are also never used (given that NIZKs are simulated).

Overall, using triangle inequality, we get that $|\Pr[\mathsf{G}_0^b = 1] - \Pr[\mathsf{G}_5^b = 1]| \leq \mu(\kappa)$ for some negligible function μ.

In G_5^b, the view of $\mathcal{A}$ is independent of b, i.e. $\Pr[\mathsf{G}_5^0 = 1] = \Pr[\mathsf{G}_5^1 = 1]$. For this, note that the only times b affects G_5^b are:

- During $\mathsf{OBTCHAL}_b$ for selecting $\mathsf{USK}[\mathsf{uid}_b]$, whose value is ignored.
- During $\mathsf{OBTCHAL}_b$ for selecting $\mathsf{CRD}[\boldsymbol{cid}_b^*]$, which is only used to compute f_{is}, whose value is independent of b by nontriviality check in $\mathsf{OBTCHAL}_b$,
- During $\mathsf{OBTCHAL}_b$ for maintaining the CCRD list, which $\mathcal{A}$ does not get to see.
- During SIGN, the value $\mathsf{uid} = \mathsf{OWN}[\boldsymbol{cid}]$ for a vector $\boldsymbol{cid}$ of challenge credential IDs $cid \in \mathsf{CCRD}$ depends on b. However, note that the checks on uid and $\mathsf{UPK}[\mathsf{uid}]$ are symmetric, i.e. whether or not $\perp$ is returned is independent of b. Other than those checks, UPK is only used to compute y_{ev} and y_{op}, whose values the nontriviality conditions ensure are independent of b, too (except if the opener is corrupted and a challenge credential is used, i.e. $\mathsf{oid} \in \mathsf{CO}$ and $cid_1 \in \mathsf{CCRD}$, then y_{op} is not even read).

Overall, we get

$$\begin{aligned}
&|\Pr[\mathsf{Exp}^{\text{iss-anon-0}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{iss-anon-1}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]| \\
=&|\Pr[\mathsf{G}_0^0 = 1] - \Pr[\mathsf{G}_0^1 = 1]| \\
=&|\Pr[\mathsf{G}_0^0 = 1] \underline{- \Pr[\mathsf{G}_5^0 = 1] + \Pr[\mathsf{G}_5^0 = 1]} - \Pr[\mathsf{G}_0^1 = 1]| \\
=&|\Pr[\mathsf{G}_0^0 = 1] - \Pr[\mathsf{G}_5^0 = 1] + \Pr[\underline{\mathsf{G}_5^1} = 1] - \Pr[\mathsf{G}_0^1 = 1]| \\
\leq&|\Pr[\mathsf{G}_0^0 = 1] - \Pr[\mathsf{G}_5^0 = 1]| + |\Pr[\mathsf{G}_5^1 = 1] - \Pr[\mathsf{G}_0^1 = 1]| \\
\leq&2 \cdot \mu(\kappa)
\end{aligned}$$

so $|\Pr[\mathsf{Exp}^{\text{iss-anon-0}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{iss-anon-1}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]|$ is negligible for all $\mathcal{A}$ as required. ■

Theorem 2 (Signature anonymity of Π_{UAS})**.** *If the NIZK system is zero-knowledge and simulation extractable, and the public-key encryption scheme is correct and IND-CPA secure, then* Π_{UAS} *satisfies signature anonymity as defined in Definition 3.*

Theorem 2. Consider the following sequence of games.

$\mathsf{G}_0^b = \mathsf{Exp}^{\text{sig-anon-}b}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$ is the original game from Definition 3.

G_1^b works like G_0^b, except that for each $(\cdot, \mathsf{oid}, \cdot, \cdot, \Sigma) \in \mathsf{SIG}$ created during SIGN queries, we remember the signature's corresponding y_{op} value, and use it during $\mathsf{OPEN}(\mathsf{oid}, \cdot, \Sigma, \cdot, \cdot)$ queries instead of of decrypting c_{op} (as in Theorem 1's G_1^b). It is easy to see that correctness of the encryption scheme implies $\Pr[\mathsf{G}_0^b = 1] = \Pr[\mathsf{G}_1^b = 1]$.

G_2^b works like G_1^b, except that Setup and replaced with $\mathsf{SimSetup}$ and the NIZK proofs are simulated (analogous to G_2^b in the proof of Theorem 1 but also replacing the call to NIZK.Prove$^{\mathcal{R}_{\text{ev}}}$ in the $\mathsf{SIGCHAL}_b$ oracle).
Similarly to Theorem 1, through straightforward reductions, one can show that $|\Pr[\mathsf{G}_1^b = 1] - \Pr[\mathsf{G}_2^b = 1]| \leq \sum_{i=1}^{3} |\Pr[\mathsf{Exp}^{\text{zk-0}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{zk-1}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1]|$ is negligible.

G_3^b works like G_2^b, except that during OPEN queries, whenever we would compute $y_{\text{op}} = \text{E.Dec}(osk, c_{\text{op}})$, G_3^b instead computes y_{op} as $(\cdot, \cdot, \cdot, y_{\text{op}}, r) = \text{NIZK.}\mathsf{Extract}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, \tau_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), \pi_{\text{ev}})$.
Similarly to Theorem 1, one can show that $|\Pr[\mathsf{G}_2^b = 1] - \Pr[\mathsf{G}_3^b = 1]| \leq \Pr[\mathsf{Exp}^{\text{simext}}_{\text{NIZK},\mathcal{B}}(\kappa) = 1]$ is negligible.

G_4^b works like G_3^b, except that during $\mathsf{SIGCHAL}_b$ queries, if $\mathsf{oid} \notin \mathsf{CO}$, then instead of $c_{\text{op}} \leftarrow \text{E.Enc}(ek, y_{\text{op}}; r)$, the oracle computes $c_{\text{op}} \leftarrow \text{E.Enc}(ek, 0; r)$ (for some fixed message "0" in the encryption scheme's message space).
Similarly to Theorem 1, but applied to $\mathsf{SIGCHAL}_b$ instead of SIGN, one can show that $|\Pr[\mathsf{G}_3^b = 1] - \Pr[\mathsf{G}_4^b = 1]| \leq p(\kappa) \cdot |\Pr[\mathsf{Exp}^{\text{IND-CPA-0}}_{\text{E},\mathcal{B}}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{IND-CPA-1}}_{\text{E},\mathcal{B}}(\kappa) = 1]|$, where p is a polynomial upper-bound for the number of honest openers created by $\mathcal{A}$, is negligible.

Overall, using triangle inequality, we get that $|\Pr[\mathsf{G}_0^b = 1] - \Pr[\mathsf{G}_4^b = 1]| \leq \mu(\kappa)$ for some negligible function μ.

In G_4^b, the view of $\mathcal{A}$ is independent of b, i.e. $\Pr[\mathsf{G}_4^0 = 1] = \Pr[\mathsf{G}_4^1 = 1]$. For this, note that the only time b potentially affects G_4^b is:

- During $\mathsf{SIGCHAL}_b$ when selecting $\mathsf{USK}[\mathsf{uid}_b^*]$ and $\mathsf{CRD}[\boldsymbol{cid}_b^*]$ as input to Sign. However, $\mathsf{USK}[\mathsf{uid}_b^*]$ and $\mathsf{CRD}[\boldsymbol{cid}_b^*]$ are never used in Sign (because the proof is simulated) except to compute y_{ev} and y_{op} with the corresponding $\mathsf{UPK}[\mathsf{uid}_b^*]$. Note that y_{ev} is independent of b because of the nontriviality check in $\mathsf{SIGCHAL}_b$. If $\mathsf{oid} \in \mathsf{CU}$, the same holds for y_{op}, if $\mathsf{oid} \notin \mathsf{CU}$, y_{op} is not even read during Sign (as defined in G_4^b). Hence overall, the execution of $\mathsf{SIGCHAL}_b$, from $\mathcal{A}$'s view, is independent of b.

Overall, with the same argument as for Theorem 1, we get that $|\Pr[\mathsf{Exp}^{\text{sig-anon-0}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{sig-anon-1}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1]|$ is negligible for all $\mathcal{A}$ as required. ■

Theorem 3 (Issuance unforgeability of Π_{UAS})**.** *If the* NIZK *scheme is extraction zero-knowledge and simulation extractable, and the* SBCM *scheme is correct, unforgeable, and has deterministically derived public keys, then* Π_{UAS} *satisfies issuance unforgeability as defined in Definition 4.*

Theorem 3. Let $\mathcal{A}$ be a PPT adversary. Consider the following sequence of games.

$\mathsf{G}_0 = \mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa)$ is the original game from Definition 4.

G_1 works like G_0 except that

- Invocations of $\pi \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}), \cdot)$ in the OBTAIN oracle are replaced with $\pi \leftarrow \text{NIZK.}\mathsf{Sim}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, \tau_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}))$. We do not change the OBTISS oracle.
- Invocations of $\pi_{\text{ev}} \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), \cdot)$ are replaced with $\pi_{\text{ev}} \leftarrow \text{NIZK.}\mathsf{Sim}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, \tau_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek))$ (this happens in the SIGN oracle).

Through two straightforward reductions $\mathcal{B}_1, \mathcal{B}_2$ (in a hybrid fashion, incrementally replacing the proofs/setup for each of the two relations), one can show that $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| \leq \sum_{i=1}^{2} |\Pr[\mathsf{Exp}^{\text{extzk-0}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{extzk-1}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1]|$ is negligible. For that reduction, note

that by design of the games, whenever we invoke NIZK.Prove, the witness used is valid. For the other nontriviality condition of extraction zero-knowledge (namely that we must not ask the extraction oracle to extract from a proof we queried): For $\mathcal{R}_{\text{ev}}$, we never extract (indeed, the standard zero-knowledge property suffices there). For $\mathcal{R}_{\text{is}}$, we call ExtIss to decide the winning conditions, cf. Figure 12, but only if $\mathsf{ISR}[cid] \in \mathsf{HI}$, implying that the issuance of cid happened outside of the OBTAIN oracle, so the corresponding issuance proof for cid was not simulated.

G_2 works like G_1 except that when an honest issuer issues to an honest user, we lazily execute this protocol only when the user is corrupted. This means that

- In the OBTISS oracle for user uid, we defer computation of the user's NIZK proof π, the intermediate signature β, and the final signature $crd = \sigma$. We just put $\pi = \bot$ and $crd = \bot$ into the $\mathbf{reg}[cid]$ and $\mathsf{CRD}[cid]$ datastrutures, respectively. These values are never read while $\mathsf{uid} \in \mathsf{HU}$ (the changes in G_1 imply that crd is not read in SIGN or OBTAIN, and π within the transcript $\mathbf{reg}[cid]$ is only read when deciding the winning condition if $\mathsf{uid} \notin \mathsf{HU}$).
- When $\mathsf{UCORR}(\mathsf{uid})$ or $\mathsf{ICORR}(\mathsf{uid})$ is called, all deferred computations for that user take place in the intended order. After each computation, we plug in the new values of π into $\mathbf{reg}[cid]$ and the new value of crd into $\mathsf{CRD}[cid]$ as appropriate (the next deferred computation may depend on crd).

There is no difference in the results of G_2 and G_1 (i.e. $\Pr[\mathsf{G}_1 = 1] = \Pr[\mathsf{G}_2 = 1]$), given that crd, reg are never read until the user is corrupted.

G_3 works like G_2 except that we add a NIZK extractor call to the ISSUE oracle, i.e. after the Iss protocol verifies the issuance proof π, we additionally run $(\cdot, usk, \cdot, r) \leftarrow \text{NIZK.}\mathsf{Extract}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, \tau_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}), \pi)$ (using the appropriate variable values from the context). The value is computed but unused (will be part of the reduction to SBCM unforgeability), so there is no difference between G_3 and G_2 in output distribution, i.e. $\Pr[\mathsf{G}_1 = 1] = \Pr[\mathsf{G}_2 = 1]$.

Overall, $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_3 = 1]|$ is negligible, meaning that if we show that $\Pr[\mathsf{G}_3 = 1]$ is negligible, then also $\Pr[\mathsf{G}_0 = 1]$ is negligible. So in the following, we consider G_3.

We define three events for G_3:

$\mathsf{FailRepeat}$ is the event that during ISSUE,OBTISS, the cid^{I} chosen by the issuer has already been used, i.e. $(cid^{\text{I}}, \cdot) \in \mathsf{CRD}$.

$\mathsf{FailExtr}$ is the event that $\neg\mathsf{FailRepeat}$ but some NIZK.$\mathsf{Extract}$ call (in ISSUE and in ExtIss) does not return a valid witness.

$\mathsf{FailForge}$ is the event that $\neg\mathsf{FailRepeat} \wedge \neg\mathsf{FailExtr}$, but G_3 outputs 1.

Because the honest issuer chooses cid^{I} from a superpoly domain, and there are at most polynomially many entries in CRD, the event $\mathsf{FailRepeat}$ occurs only with negligible probability. Through a straightforward reduction to the simulation extractability property of NIZK, which outputs an unextractable proof whenever $\mathsf{FailExtr}$ happens, one can show that $\Pr[\mathsf{FailExtr}]$ is negligible. For that reduction, note that we only call NIZK.$\mathsf{Extract}$ on non-simulated proofs, as argued in the

following. First, we never extract for $\mathcal{R}_{\text{ev}}$. Consider the following cases where extraction of proof π for $\mathcal{R}_{\text{is}}$ with statement $x = (\cdot, \cdot, cid = (cid^{\text{I}}, cid^{\text{U}}), \cdot, \cdot, \cdot)$ could potentially fail:

- If $cid \notin \mathsf{CRD}$, then we clearly did not simulate any statement involving cid.
- If $\mathsf{CRD}[cid]$ was written during OBTISS (i.e. the proof is lazily computed during $\mathsf{UCORR}/\mathsf{ICORR}$), then the corresponding proof π was not simulated (because we do not simulate those proofs).
- If $\mathsf{CRD}[cid]$ was written during ISSUE, then due to $\neg\mathsf{FailRepeat}$, no proof containing the same cid^{I} has been simulated at the point where $\mathcal{A}$ output π (and hence deterministic extraction of π must succeed).
- If $\mathsf{CRD}[cid]$ was written during OBTAIN, then the corresponding issuer is corrupt ($\mathsf{iid} \in \mathsf{CI}$), so we never attempt ExtIss on cid.

Finally, consider the event $\mathsf{FailForge}$. We show via reduction to SBCM unforgeability that $\Pr[\mathsf{FailForge}]$ is negligible. For this, let p be a (polynomial) upper bound on the number of honest users that $\mathcal{A}$ creates (i.e. number of HUGEN calls). We construct $\mathcal{B}(par_{\text{SBCM}}, vk_{\text{SBCM}})$ against SBCM unforgeability that runs G_3 with a few modifications:

- $\mathcal{B}$ uses par_{SBCM} from its input instead of generating it itself like G_3.
- $\mathcal{B}$ chooses a random index $i^* \leftarrow \{1, \ldots, p(\kappa)\}$ and for $\mathsf{HUGEN}(\mathsf{uid}^*)$ queries, if this is the i^*th HUGEN query, then $\mathcal{B}$ sets $\mathsf{UK}[\mathsf{uid}^*] = (ipk^*, isk^*) = (vk_{\text{SBCM}}, \bot)$ (i.e. we embed the challenge verification key as the user's key, we are now missing isk^*).
- If user i^* gets corrupted, $\mathcal{B}$ halts.
- Whenever G_3 would run $\text{SBCM.Sign}(isk, c, (cid, \boldsymbol{a}))$ (which is during ISSUE or when $\mathsf{UCORR}/\mathsf{ICORR}$ runs the deferred protocols from OBTISS, cf. G_2), then $\mathcal{B}$ instead calls its oracle $\mathsf{SIGN}(usk, (cid, \boldsymbol{a}), r)$, using usk, r from the context (for $\mathsf{UCORR}/\mathsf{ICORR}/\mathsf{OBTISS}$, the values usk, r are available because the user is honest, for ISSUE, the values are output by the NIZK extractor, see G_3).
- Furthermore, when $\mathsf{UCORR}/\mathsf{ICORR}$ runs the deferred protocols from OBTISS using $upk, usk, r, cid, \boldsymbol{a}$ with issuer i^*, after querying $\sigma \leftarrow \mathsf{SIGN}(usk, (cid, \boldsymbol{a}), r)$, the reduction $\mathcal{B}$ also computes $(upk', usk', \cdot, r') = \text{NIZK.Extract}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, \tau_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}), \pi)$ from the proof π created during the Obt protocol. If $usk \neq usk'$ and $c = \text{SBCM.Blind}(ipk^*, usk, (cid, \boldsymbol{a}), r) = \text{SBCM.Blind}(ipk^*, usk', (cid, \boldsymbol{a}), r')$, then $\mathcal{B}$ computes $\sigma^* = \text{SBCM.Unblind}(ipk^*, \beta, c, r', usk', (cid, \boldsymbol{a}))$ using the partial signature β returned by the SIGN oracle query, halts and outputs forgery $(\sigma^*, \overline{\boldsymbol{msg}}^*, \boldsymbol{msg}^*) = (\sigma^*, usk', (cid, \boldsymbol{a}))$.
- If $\mathsf{FailForge}$ occurs during $\mathsf{CheckEndorsementCreds}$ when checking index i, then $\mathcal{B}$ checks that $\mathsf{IPK}[\mathsf{ISR}[cid_i]] = ipk^*$ (if not, it halts without outputting a forgery). Then $\mathcal{B}$ takes the witness $(upk, usk, \{(cid_j, \boldsymbol{a}_j, crd_j)\}_{j\in[n]}, r)$ extracted in line 11 of $\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$ and outputs the forgery $(\sigma^*, \overline{\boldsymbol{msg}}^*, \boldsymbol{msg}^*) = (crd_i, usk, (cid_i, \boldsymbol{a}_i))$.

Note that we have no access to $\mathsf{USK}[\mathsf{uid}^*]$, but that value is never used, thanks to the modifications in G_1 and because we are using the oracle to issue blind signatures. If $\mathsf{FailForge}$ occurs and we guessed i^* correctly, $\mathcal{B}$ will have perfectly simulated G_3 from the point of view of $\mathcal{A}$.

We now argue that whenever FailForge occurs when checking index i such that $\mathsf{IPK}[\mathsf{ISR}[cid_i]] = ipk^*$, then $\mathcal{B}$ outputs (or has already output) a forgery. Because we guess the user, for which the FailForge event is supposed to happen, independently of the event, this will imply that $\Pr[\mathsf{FailForge}] \leq \Pr[\mathsf{Exp}^{\mathrm{EUF}}_{\mathrm{SBCM},\mathcal{B}}(\kappa) = 1] \cdot p(\kappa)$ is negligible. Note that FailForge cannot occur in line 13 of $\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$ because that would imply FailExtr, so FailForge can only occur within CheckEndorsementCreds.

First, we make the following useful observations: Whenever FailForge occurs, the signature-message pair output by $\mathcal{B}$ is valid (i.e. it survives the signature verification of the challenger). If $\mathcal{B}$ outputs the forgery during CheckEndorsementCreds, this is because FailForge implies ¬FailExtr, so the extracted signature definitely verifies correctly. If $\mathcal{B}$ outputs the forgery during UCORR or ICORR, then it follows from correctness of the SBCM scheme that the opening to the commitment to usk' also unblinds the partial signature β to a valid signature. As a second observation, whenever FailForge occurs, for every possible value cid, $\mathcal{B}$ has made at most one signature query of the form $\mathsf{SIGN}(\cdot, (cid, \cdot), r)$. This is because FailForge by definition implies ¬FailRepeat.

Armed with those insights, let i be the index checked by CheckEndorsementCreds when FailForge occurs. Consider the following case distinction for when FailForge occurs. We argue that in each of those scenarios, $\mathcal{B}$ has not queried for a signature on the forgery message.

- If FailForge occurs because $cid_i \notin \mathsf{CRD}$, then $\mathcal{B}$ has never queried a signature involving cid_i.
- If FailForge occurs because of the condition $\mathsf{ISR}[cid_i] \neq \mathsf{iid}_i$ in line 3 of CheckEndorsementCreds. Because of ¬FailRepeat, this means that the signature involving cid_i has been created honestly by $\mathcal{B}$ without involving the signature oracle for $ipk^* = \mathsf{ISR}[cid_i]$, i.e. $\mathcal{B}$ has not queried for a message involving cid_i.
- If FailForge occurs because $\mathsf{OWN}[cid_i] \in \mathsf{HU}$ in line 5 of CheckEndorsementCreds (but $cid_i \in \mathsf{CRD}$), then cid_i has been chosen during OBTISS. Because of ¬FailRepeat, we have not asked for a signature containing cid_i during ISSUE. Because $\mathsf{OWN}[cid_i] \in \mathsf{HU}$, we have not (yet) queried for the signature containing cid_i during UCORR/ICORR. Hence we have not queried for a signature containing cid_i at all.
- If FailForge occurs because $\mathsf{ATT}[cid_i] \neq \boldsymbol{a}_i$ in line 3 of CheckEndorsementCreds (but $cid_i \in \mathsf{CRD}$), then when we (potentially) queried for the signature containing cid_i, we would have done so with attributes $\mathsf{ATT}[cid_i]$. Because $\mathsf{ATT}[cid_i] \neq \boldsymbol{a}_i$, we have never requested a signature containing $(cid_i, \boldsymbol{a}_i)$.
- If FailForge occurs because $upk \neq upk'$ in line 9 of CheckEndorsementCreds, then there are two cases how $\mathcal{B}$ could have potentially queried for the forgery message:
  - Assume we have queried a signature on $(usk', (cid_i, \boldsymbol{a}_i))$ during ISSUE (where usk' is output by $\mathrm{NIZK.Extract}^{\mathcal{R}}_{\mathrm{is}}$). Because $upk \neq upk'$ and because of deterministically derived public keys, we get $usk \neq usk'$, i.e. the (unique) message we have queried for cid_i is different from the forgery message.
  - Assume we have queried a signature on $(usk, (cid_i, \boldsymbol{a}_i))$ during UCORR/ICORR, for a to-be-corrupted user's key pair (upk, usk). In this case, because the extracted upk' differs from the user's upk, and because ¬FailExtr, $\mathcal{B}$ at the time of the UCORR/ICORR query would have also extracted, would have found $usk' \neq usk$ (implied by $upk' \neq upk$ and deterministically derived public keys), and would have output a forgery on $(usk', (cid_i, \boldsymbol{a}_i))$, having only queried $(usk, (cid_i, \boldsymbol{a}_i))$.

Overall, whenever $\mathsf{FailForge}$ occurs and $\mathcal{B}$ guessed i^* correctly, then $\mathcal{B}$ outputs a valid forgery. Hence $\Pr[\mathsf{G}_3 = 1] \leq \Pr[\mathsf{FailRepeat}] + \Pr[\mathsf{FailExtr}] + \Pr[\mathsf{FailForge}]$ is negligible. ■

Theorem 4 (Signature unforgeability of Π_{UAS})**.** *If the underlying NIZK scheme is complete, extraction zero-knowledge and simulation extractable, the public key encryption scheme is correct, and the* SBCM *scheme is correct, unforgeable, and has deterministically derived public keys, then* Π_{UAS} *satisfies signing unforgeability as defined in Definition 5.*

Proof. Most of this proof is analogous to the proof of Theorem 3, as both games use the same set of oracles, and they share the checks made by $\mathsf{CheckEndorsementCreds}$.

Specifically, we define $\mathsf{G}_0, \ldots, \mathsf{G}_3$ analogously to the proof of Theorem 3 (but as modifications of $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ rather than $\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$). We can conclude that $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_3 = 1]|$ is negligible using the arguments from the other proof. The only nontrivial difference here is that in $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$, we *do* extract from proofs of the relation $\mathcal{R}_{\text{ev}}$ (as part of running $\mathsf{ExtSign}$ to decide the winning condition), so for the game-hop from G_0 to G_1, we need to additionally argue that we do not try to extract from a simulated $\mathcal{R}_{\text{ev}}$ proof. However, this is easy to see and follows directly from the check in line 6 of $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$.

Then, analogous to Theorem 3, we define error events for G_3:

$\mathsf{FailRepeat}$ is the event that during ISSUE,OBTISS, the cid^{I} chosen by the issuer has already been used, i.e. $(cid^{\text{I}}, \cdot) \in \mathsf{CRD}$.

$\mathsf{FailExtr}$ is the event that $\neg\mathsf{FailRepeat}$ but some NIZK.$\mathsf{Extract}$ call (in ISSUE and in ExtIss) does not return a valid witness.

$\mathsf{FailJudge}$ is the event that $\neg\mathsf{FailExtr}$ but G_3 outputs 1 in line 17/18(cf. Figure 12) because the honestly produced (y_{op}, π) pair at line 15 is inconsistent to $f_{\text{op}}(\ldots)$.

$\mathsf{FailForge}$ is the event that $\neg\mathsf{FailRepeat} \wedge \neg\mathsf{FailExtr} \wedge \neg\mathsf{FailJudge}$, but G_3 outputs 1.

In contrast to that proof, we have added another error event $\mathsf{FailJudge}$ here, but the other events are perfectly analogous.

As in Theorem 3, we can argue that $\mathsf{FailRepeat}$ and $\mathsf{FailExtr}$ only occur with negligible probability.

We can show that $\Pr[\mathsf{FailJudge}]$ is negligible through a straightforward reduction $\mathcal{B}$ to the soundness property of the NIZK. If $\mathsf{FailJudge}$ occurs, let $y'_{\text{op}} = f_{\text{op}}(upk, \{(cid_i, \boldsymbol{a}_i)\}, m) \neq y_{\text{op}}$. In that scenario, we have found $\mathsf{oid}, r, \sigma = (\pi_{\text{ev}}, c_{\text{op}}), y_{\text{op}}, \pi_{\text{op}}$ such that NIZK.$\mathsf{Verify}^{\mathcal{R}_{\text{op}}}(crs_{\text{op}}, \pi, (\mathsf{OPK}[\mathsf{oid}], c_{\text{op}}, y_{\text{op}})) = 1$ while also $c_{\text{op}} = \text{E.Enc}(\mathsf{OPK}[\mathsf{oid}], y'_{\text{op}}, r)$ as guaranteed by extraction of π_{ev} (note that $\mathsf{FailJudge}$ implies $\neg\mathsf{FailExtr}$). The second equation, which says that c_{op} is an encryption of y'_{op} contradicts the relation $\mathcal{R}_{\text{op}}$, which says that c_{op} is an encryption of $y_{\text{op}} \neq y'_{\text{op}}$. This is a contradiction because the encryption scheme is correct (and $\mathcal{R}_{\text{op}}$ also guarantees existence of a fitting osk to $\mathsf{OPK}[\mathsf{oid}]$). Hence π can be used by $\mathcal{B}$ to win the soundness game.

Finally, we show that $\Pr[\mathsf{FailForge}]$ is negligible through a reduction $\mathcal{B}$ to the unforgeability property of the SBCM scheme. $\mathcal{B}$ is constructed perfectly analogously to Theorem 3, except that when $\mathsf{FailForge}$ occurs because G_3 outputs 1 during check of index i in $\mathsf{CheckEndorsementCreds}$ and $\mathsf{IPK}[\mathsf{iid}_i] = ipk^*$ is our challenge public key (i.e. $\mathcal{B}$ has guessed the correct issuer), then $\mathcal{B}$ outputs the forgery $(\sigma^*, \overline{\boldsymbol{msg}}^*, \boldsymbol{msg}^*) = (crd_i, usk, (cid_i, \boldsymbol{a}_i))$, using the appropriate witness $(upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i\in[n]}, y_{\text{op}}, r)$ extracted from the signature in line 10 of $\mathsf{Exp}^{\text{ext-sign}}_{\mathsf{UAS},\mathcal{A}}$ (whereas the

corresponding $\mathcal{B}$ in Theorem 3 extracts from an issuance proof). The rest of the description of $\mathcal{B}$ can be taken verbatim from the proof of Theorem 3.

For the analysis, note that $\mathsf{FailForge}$ cannot occur in line 12 of $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ because that would imply $\mathsf{FailExtr}$, and it cannot occur in line 17/18 because that would imply $\mathsf{FailExtr}$ (or contradict correctness of decryption or contradict completeness of the NIZK). So overall, $\mathsf{FailForge}$ can only occur within $\mathsf{CheckEndorsementCreds}$.

Now we are in the same setting as in Theorem 3, where we can argue in the exact same way that if $\mathcal{B}$ outputs a forgery, it has not queried its signature oracle for the forgery message. We omit the details and leave the straightforward adaptation of the corresponding argumentation for Theorem 3 to the reader.

Overall, whenever $\mathsf{FailForge}$ occurs and $\mathcal{B}$ guessed i^* correctly, then $\mathcal{B}$ outputs a valid forgery. Hence $\Pr[\mathsf{G}_3 = 1] \leq \Pr[\mathsf{FailRepeat}] + \Pr[\mathsf{FailExtr}] + \Pr[\mathsf{FailJudge}] + \Pr[\mathsf{FailForge}]$ is negligible. ■

Theorem 5 (Non-frameability of Π_{UAS})**.** *If the NIZK system is extraction zero-knowledge and simulation extractable and the* SBCM *scheme is correct, blind, and unforgeable, then* Π_{UAS} *satisfies non-frameability as defined in Definition 6.*

Proof. Let $\mathcal{A}$ be a PPT adversary against Π_{UAS} non-frameability. Consider the following modified games:

G_0 works like $\mathsf{Exp}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}$.

G_1 works like G_0 except for two changes:

- After the Iss protocol during $\mathsf{OBTISS}, \mathsf{ISSUE}$ verifies the issuance proof π, we additionally run $(\cdot, usk, \cdot, r) \leftarrow \text{NIZK.}\mathsf{Extract}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, \tau_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}), \pi)$ (using the appropriate variable values from the context). The result will be later used in the SBCM unforgeability reduction. If the extracted witness is invalid, G_1 halts and returns 0.
- We replace the old winning condition check "if $f_{\text{op}}(upk, \{(cid_i, \boldsymbol{a}_i)\}, m) \neq y_{\text{op}}$, then return 1" (line 12 of $\mathsf{Exp}^{\text{frame}}_{\mathsf{UAS},\mathcal{A}}$) with a new "if the witness $(upk, usk, \{(cid_i, \boldsymbol{a}_i, crd_i)\}_{i\in[n]}, r)$ extracted by $\text{NIZK.}\mathsf{Extract}^{\mathcal{R}_{\text{ev}}}$ in $\mathsf{ExtSign}$ is invalid, then return 0" check.

Through a straightforward reduction to the simulation-extractability property (note: a proof of knowledge property would suffice here) of the NIZK, one can show that the probability that the new losing condition happens is negligible. Note that we do not simulate any proofs in G_0. Furthermore, whenever the new losing condition from the second bullet point does not happen, then also the old winning condition does not happen (because correct extraction directly implies the old equation), so the overall change can only be detected with negligible probability. Hence $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| \leq \Pr[\mathsf{Exp}^{\text{simext}}_{\text{NIZK},\mathcal{B}} = 1]$ is negligible

G_2 works like G_1 except that

- Invocations of $\pi \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}), \cdot)$ are replaced with $\pi \leftarrow \text{NIZK.}\mathsf{Sim}^{\mathcal{R}_{\text{is}}}(crs_{\text{is}}, \tau_{\text{is}}, (f_{\text{is}}, c, cid, \boldsymbol{a}, ipk, \{ipk_i\}_{i\in[n]}, y_{\text{is}}))$ (this happens in the OBTAIN and OBTISS oracles).
- Invocations of $\pi_{\text{ev}} \leftarrow \text{NIZK.Prove}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek), \cdot)$ are replaced with $\pi_{\text{ev}} \leftarrow \text{NIZK.}\mathsf{Sim}^{\mathcal{R}_{\text{ev}}}(crs_{\text{ev}}, \tau_{\text{ev}}, (m, f_{\text{ev}}, y_{\text{ev}}, f_{\text{op}}, c_{\text{op}}, \{ipk_i\}_{i\in[n]}, ek))$ (this happens in the SIGN oracle).

Through two straightforward reductions $\mathcal{B}_1, \mathcal{B}_2$ (in a hybrid fashion, incrementally replacing the proofs/setup for each of the two relations), one can show that $|\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]| \leq \sum_{i=1}^{2} |\Pr[\mathsf{Exp}^{\text{extzk-0}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1] - \Pr[\mathsf{Exp}^{\text{extzk-1}}_{\text{NIZK},\mathcal{B}_i}(\kappa) = 1]|$ is negligible. For that reduction, note that by design of the games, whenever we invoke NIZK.Prove, the witness used is valid. For the other nontriviality condition of extraction zero-knowledge (namely that we must not ask the extraction oracle to extract from a proof we queried): For $\mathcal{R}_{\text{is}}$, we never extract (indeed, the standard zero-knowledge property suffices there). For $\mathcal{R}_{\text{ev}}$, we call $\mathsf{ExtSign}$ to decide the winning conditions, cf. Figure 12, but only if there is no entry $(\cdot, \cdot, m, f_{\text{ev}}, \Sigma) \in \mathsf{SIG}$, implying that the corresponding signature statement-proof pair was not simulated.

G_3 works like G_2 except that G_3 chooses a random index $i^* \leftarrow \{1, \ldots, p(\kappa)\}$, where $p(\kappa)$ is a polynomial upper bound on the number of honest users $\mathcal{A}$ creates (i.e. number of HUGEN calls). Furthermore, let uid^* denote the ID of the i^*th user (or $\perp$ if that user was never created). G_3 changes the check win condition "if $\exists \mathsf{uid} \in \mathsf{HU} s.t. \mathsf{UPK}[\mathsf{uid}] = upk$, then return 1" to "if $\mathsf{uid}^* \in \mathsf{HU} \wedge \mathsf{UPK}[\mathsf{uid}^*] = upk$, then return 1". It is easy to see that $\Pr[\mathsf{G}_3 = 1] \geq \Pr[\mathsf{G}_2 = 1]/p(\kappa)$. In particular, if we can show that $\Pr[\mathsf{G}_3 = 1]$ is negligible, then necessarily, $\Pr[\mathsf{G}_2 = 1]$ is also negligible (which in turn implies that $\Pr[\mathsf{G}_0 = 1]$ is negligible).

G_4 works like G_3 except that if uid^* is corrupted, G_4 just aborts and outputs 0. We have $\Pr[\mathsf{G}_3 = 1] = \Pr[\mathsf{G}_4 = 1]$ because after uid^* gets corrupted, G_3 will never output 1.

G_5 works like G_4 except that it generates some dummy user keys $(upk_{\text{dummy}}, usk_{\text{dummy}}) \leftarrow \text{UKG}(\kappa)$ at the beginning of the experiment. Whenever $\mathcal{A}$ queries $\mathsf{OBTAIN}(\mathsf{uid}^*, \cdot, \cdot, \cdot)$ or $\mathsf{OBTISS}(\mathsf{uid}^*, \cdot, \cdot, \cdot)$ (where uid^* is the guessed user as in G_4), then G_5 uses $(upk_{\text{dummy}}, usk_{\text{dummy}})$ for that query instead of $(\mathsf{UPK}[\mathsf{uid}^*], \mathsf{USK}[\mathsf{uid}^*])$ to receive the credential. Note that $\mathsf{USK}[\mathsf{uid}^*]$ was only used in G_4 in these queries to commit to $\mathsf{USK}[\mathsf{uid}^*]$ during the SBCM blind signing protocol (the proofs are simulated and hence independent of $\mathsf{USK}[\mathsf{uid}^*]$). The resulting credential in G_5 will be a signature on the wrong key usk^*, but that signature is not actually ever used anywhere (cf. G_2 and G_4).

Through a straightforward reduction $\mathcal{B}$ to the SBCM blindness property, one can show that $|\Pr[\mathsf{G}_4 = 1] - \Pr[\mathsf{G}_5 = 1]| = |\Pr[\mathsf{Exp}^{\text{blind-1}}_{\text{SBCM},\mathcal{B}}] - \Pr[\mathsf{Exp}^{\text{blind-0}}_{\text{SBCM},\mathcal{B}}]|$ is negligible.

In G_5, the value $\mathsf{USK}[\mathsf{uid}^*]$ is only used for SBCM.Sign operations (when uid^* acts as an issuer), but nowhere else anymore.

Finally, we show that $\Pr[\mathsf{G}_5 = 1]$ is negligible by reduction to SBCM unforgeability. Intuitively, the only way to win G_5 is to for the extractor to output a valid secret key usk^* for the guessed user upk^*. Because user keys are SBCM keys, being able to compute usk^* breaks SBCM unforgeability.

More formally, define adversary $\mathcal{B}^{\mathsf{SIGN}}(par_{\text{SBCM}}, upk^*)$ against the SBCM unforgeability game $\mathsf{Exp}^{\text{EUF}}_{\text{SBCM},\mathcal{B}}$. $\mathcal{B}$ runs G_5, but uses par_{SBCM} from its input and embeds upk^* as $\mathsf{UPK}[\mathsf{uid}^*]$ (where uid^* is the guessed user in G_5). Whenever $\mathsf{USK}[\mathsf{uid}^*]$ would be used in G_5 (which is only for SBCM.Sign operations), $\mathcal{B}$ instead uses its SIGN oracle (using the appropriate extracted values established in G_1). When G_5 outputs 1, $\mathcal{B}$ chooses some $\overline{\boldsymbol{msg}}, \boldsymbol{msg}$ that it has not queried before, and some random r, then takes the extracted usk^* to compute $c \leftarrow \text{SBCM.Blind}(upk^*, \overline{\boldsymbol{msg}}, \boldsymbol{msg}, r)$ and $\sigma = \text{SBCM.Unblind}(upk^*, \text{SBCM.Sign}(usk^*, c, \boldsymbol{msg}), c, r, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$. It outputs the forgery $(\sigma, \overline{\boldsymbol{msg}}, \boldsymbol{msg})$. By correctness of the SBCM scheme, σ is a valid signature on $\overline{\boldsymbol{msg}}, \boldsymbol{msg}$, hence a forgery.

Overall, whenever G_5 outputs 1, $\mathcal{B}$ outputs an SBCM forgery. Hence $\Pr[\mathsf{G}_5 = 1] \leq \Pr[\mathsf{Exp}^{\text{EUF}}_{\text{SBCM},\mathcal{B}} = 1]$ is negligible. As argued above, this implies that $\Pr[\mathsf{G}_0 = 1]$ is negligible. ■

D Models and Proofs for Relationships with Other Schemes

D.1 Digital Signatures

For digital signatures, we follow the conventional EUF-CMA security model [GMR88], whose unforgeability property we reproduce in Fig. 18 for self-containedness. A digital signature scheme ds is EUF-CMA unforgeable if for all ppt $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}(1^\kappa) = 1]$ is negligible.

$\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}(1^\kappa)$

1 : $par \leftarrow \text{Setup}(1^\kappa)$
2 : $(upk, usk) \leftarrow \text{KG}(par)$
3 : $(m, \sigma) \leftarrow \mathcal{A}^{\text{Sign}(usk,\cdot)}(upk)$
4 : **if** $\mathcal{A}$ queried Sign on m : **return** 0
5 : **return** $\text{Verify}(upk, m, \sigma)$

Figure 18: Security game for EUF-CMA signatures [GMR88].

Security of $\Pi^{\text{ds}}_{\mathsf{UAS}}$. We now prove that $\Pi^{\text{ds}}_{\mathsf{UAS}}$, the $(\cdot, f^{upk}_{\text{ev}}, f^{0}_{\text{op}})$-$\Pi_{\mathsf{UAS}}$ restriction specified in Section 6.1, is a secure digital signature scheme.

Theorem 7 (Unforgeability of $\Pi^{\text{ds}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction (Section 6.1) has signature unforgeability and non-frameability according to Definition 5 and Definition 6, then $\Pi^{\text{ds}}_{\mathsf{UAS}}$ satisfies EUF-CMA unforgeability as defined in Fig. 18.*

The proof is ultimately a reduction to the non-frameability property of UAS. However, before we can start that reduction, we need two intermediate arguments. First, because the adversary against non-frameability is expected to output a valid opening proof, we first add the honest computation of an opening proof to the game and argue that it's unlikely to fail. Second, because UAS are anonymous by default, non-frameability's winning condition uses the signer key *upk* output *by the extractor* $\mathsf{ExtSign}$. We need to ensure that *upk* extracted by $\mathsf{ExtSign}$ is indeed the challenge *upk* from the UF-CMA game (which *should* be the case for good $\mathsf{ExtSign}$, as $f^{\mathbf{upk}}_{\text{op}}(\dots) = upk$ by construction).

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}(1^\kappa)$ for our $\Pi^{\text{ds}}_{\mathsf{UAS}}$ construction. Now, consider the following sequence of games:

G_0 This is the standard $\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}(1^\kappa)$.

G_1 Behaves like G_0 but it computes par' as $(par', \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$ (note that the distribution of par' does not change). Furthermore, when $\mathcal{A}$ outputs a winning forgery (m, σ):

- G_1 runs the UAS extractor $(upk', \cdot) \leftarrow \mathsf{ExtSign}(\tau, \Sigma)$ with the appropriate $\Sigma = (\sigma, upk)$ (where *upk* is the challenge public key used in $\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}$). If the extracted upk' is different from *upk*, then G_1 outputs 0. We call the event that this happens `FailEv`.
- G_1 computes an opening proof $(y_{\text{op}}, \pi_{\text{op}}) \leftarrow \text{Open}(osk, \emptyset, \Sigma, m, f^{\mathbf{upk}}_{\text{ev}})$ for the signature output by $\mathcal{A}$. If π_{op} is invalid, i.e. $y_{\text{op}} \neq 0$ or $\text{Judge}(opk, \emptyset, y_{\text{op}}, \pi_{\text{op}}, \Sigma, m, f^{\mathbf{upk}}_{\text{ev}}) = 0$ (where *osk* is as computed during $\Pi^{\text{ds}}_{\mathsf{UAS}}$.Setup), then G_1 outputs 0. We call the event that this happens `FailOp`.

Clearly, there is no difference between G_0 and G_1 unless one of the two error events happens, hence $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| \leq \Pr[\texttt{FailEv} \vee \texttt{FailOp}]$ (via difference lemma).

We argue that the signature unforgeability property of UAS ensures that $\Pr[\texttt{FailEv} \vee \texttt{FailOp}]$ is negligible. In the following, we consider an adversary $\mathcal{B}$ against $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$, leveraging $\mathcal{A}$ against $\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}$. $\mathcal{B}$ uses its oracles to set up one honest user. Let uid, upk be that honest user's ID and public key, respectively. $\mathcal{B}$ uses that user's upk when simulating $\mathsf{Exp}^{\text{euf-cma}}_{\mathcal{A},\text{ds}}$ for $\mathcal{A}$. It also sets up an honest opener for $f_{\text{op}} = f^0_{\text{op}}$ and subsequently uses its corresponding opk for the simulation. Let oid be that opener's ID. Whenever $\mathcal{A}$ asks for a signature, $\mathcal{B}$ uses its own UAS SIGN oracle in the natural way to answer. When $\mathcal{A}$ outputs its candidate forgery (m, σ) and wins its unforgeability game, then $\mathcal{B}$ outputs $(\mathsf{oid}, \emptyset, \Sigma = (\sigma, upk), m, f^{\texttt{upk}}_{\text{ev}})$ to $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$. It is easy to see that $\texttt{FailEv}$ and $\texttt{FailOp}$ both correspond to winning cases for $\mathcal{B}$: If $\texttt{FailEv}$ happens, then $\mathcal{B}$ wins in line 12 of $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$, if $\texttt{FailOp}$ happens, then $\mathcal{B}$ wins in line 17/18. Because $\mathcal{B}$ is a ppt and Π_{UAS} is unforgeable, $\Pr[\texttt{FailEv} \vee \texttt{FailOp}]$ must be negligible (and hence G_0 and G_1 are indistinguishable).

Now, consider $\mathcal{A}$ playing against G_1. We can build an adversary $\mathcal{B}$ that breaks non-frameability of the underlying Π_{UAS} construction. The simulation of $\mathcal{A}$'s environment is again direct. Eventually, $\mathcal{A}$ outputs a (m, σ) pair.

When that happens, $\mathcal{B}$ outputs $(\mathsf{oid}, \emptyset, \Sigma = (\sigma, upk), m, f^{\texttt{upk}}_{\text{ev}}, 0, \pi_{\text{op}})$ in its non-frameability game. If $\mathcal{A}$ wins G_1, then σ does not belong to SIG, and hence it is accepted by Verify in $\mathcal{B}$'s non-frameability game. Moreover, Judge accepts π_{op} (because $\neg\texttt{FailOp}$ is required for $\mathcal{A}$ to win G_1). Moreover, because of $\neg\texttt{FailEv}$, we also know that $\mathsf{ExtSign}$ outputs the expected upk. upk belongs to an honest user (note that $\mathcal{B}$ does not make use of the user corruption oracle. Thus, whenever $\mathcal{A}$ wins G_1, then $\mathcal{B}$ wins its non-frameability game in line 14. This implies that $\Pr[\mathsf{G}_1 = 1]$ must be negligible. Hence $\Pr[\mathsf{G}_0 = 1]$ is negligible, as required. ■

Note that the proof also works for proving *strong* unforgeability of $\Pi^{\text{ds}}_{\mathsf{UAS}}$, i.e. signatures are non-malleable.

D.2 Group Signatures

We mostly adopt the model in [BSZ05]. In this abstraction, the opener returns an index uniquely identifying the group member who created a signature, along with a correctness proof. To ease exposition, we assume without loss of generality that this index is just the public key that group members generate randomly when joining the group[3]. A group signature scheme in [BSZ05] is composed of KG, UKG, Sign, Verify, Open and Judge algorithms, and an $\langle \text{Obt}, \text{Iss} \rangle$ interactive protocol. We omit the RReg and WReg oracles in our modeling. These oracles are there to model information flow from the issuer to the opener, allowing the opener to trace users to their identities. In UAS, such information flow does not have to be explicitly modeled, as users are identified by their public key upk instead of an arbitrary "identity" that needs to be communicated. Furthermore, the UAS definitions do not allow for concurrency when issuing membership certificates, hence we also replace the [BSZ05] oracles $\mathsf{SndToI}, \mathsf{AddU}, \mathsf{SndToU}$ with UAS-style $\mathsf{ISSUE}, \mathsf{OBTISS}, \mathsf{OBTAIN}$ oracles, which are constrained to sequential execution, plus an explicit HUGEN oracle. Finally, we omit honest users in the traceability experiment, letting $\mathcal{A}$ take control of all users. This change does not change the level of security, as corruption status does not factor into the winning condition, hence $\mathcal{A}$ might as well work with only corrupted users and simulate honest ones itself if needed.

[3]The user public key is assumed to be accessible from a public table in [BSZ05], so it is easy to translate the public key into an index, if needed.

Barring these minor modifications, the oracles are similar to those of UAS. We refer to [BSZ05] for detailed descriptions. For ease of reference, we replicate the security games in Fig. 19.

$\mathsf{Exp}^{\text{anon-}b}_{\mathcal{A},\text{gs}}(1^\kappa)$

1 : $(gpk, isk, osk) \leftarrow \text{KG}(1^\kappa)$
2 : $d \leftarrow \mathcal{A}^{\mathcal{O}_{\text{anon}-b}}(gpk, isk)$
3 : **return** d

$\mathsf{Exp}^{\text{trace}}_{\mathcal{A},\text{gs}}(1^\kappa)$

1 : $(gpk, isk, osk) \leftarrow \text{KG}(1^\kappa)$
2 : $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{trace}}}(gpk, osk)$
3 : **if** $\text{Verify}(gpk, m, \sigma) = 0$: **return** 0
4 : $(upk, \tau) \leftarrow \text{Open}(gpk, osk, m, \sigma)$
5 : **if** $\text{Judge}(gpk, upk, m, \sigma, \tau) = 0$: **return** 1
6 : **if** $\mathcal{A}$ made no queries $\mathsf{ISSUE}(upk)$: **return** 1
7 : **return** 0

$\mathsf{Exp}^{\text{frame}}_{\mathcal{A},\text{gs}}(1^\kappa)$

1 : $(gpk, isk, osk) \leftarrow \text{KG}(1^\kappa)$
2 : $(m, \sigma, upk, \tau) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{frame}}}(gpk, osk, isk)$
3 : **if** $\text{Verify}(gpk, m, \sigma) = 0$: **return** 0
4 : **return** $\exists \mathsf{uid}: \ upk = \mathsf{UPK}[\mathsf{uid}] \wedge \text{Judge}(gpk, upk, m, \sigma, \tau) = 1 \wedge$
5 : $\quad \mathcal{A}$ did not query $\mathsf{UCORR}(\mathsf{uid})$ or $\mathsf{SIGN}(\mathsf{uid}, m)$

Figure 19: Security games for group signatures [BSZ05], only with some edits to ease comparison with $\Pi^{\text{gs}}_{\mathsf{UAS}}$. In particular, since we output *upk*s instead of indexes, $i = 0$ in the traceability game now reads "*upk* has not been queried to ISSUE or OBTISS". $\mathcal{O}_{\text{anon}-b} \leftarrow \{\mathsf{Ch}_b, \mathsf{OPEN}, \mathsf{OBTAIN}, \mathsf{HUGEN}, \mathsf{UCORR}, \mathsf{CUGEN}\}$, $\mathcal{O}_{\text{trace}} \leftarrow \{\mathsf{ISSUE}\}$, and $\mathcal{O}_{\text{frame}} \leftarrow \{\mathsf{OBTAIN}, \mathsf{SIGN}, \mathsf{HUGEN}, \mathsf{UCORR}, \mathsf{CUGEN}\}$.

Security of $\Pi^{\text{gs}}_{\mathsf{UAS}}$ We prove that our $\Pi^{\text{gs}}_{\mathsf{UAS}}$ construction (Section 6.1) is an anonymous, traceable, and non-frameable group signature scheme, according to the (modified) model in [BSZ05], if the underlying Π_{UAS} construction is secure.

Theorem 8 (Anonymity of $\Pi^{\text{gs}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction has signature anonymity according to Definition 3, then $\Pi^{\text{gs}}_{\mathsf{UAS}}$ (Section 6.2) is an anonymous group signature scheme.*

Proof. $\mathcal{B}$ prepares the environment for $\mathcal{A}$'s anonymity game. For this, $\mathcal{B}$ first calls its own HUGEN, HOGEN (with f^{upk}_{op}) and ISET (with f^{upk}_{is}) oracles, and sets $gpk \leftarrow (ipk, opk)$. Then, calls ICORR on the generated issuer, to obtain its *isk* and pass it to $\mathcal{A}$. To answer $\mathcal{A}$'s oracle queries, $\mathcal{B}$ leverages its own oracles in the natural way. In particular, calls to Ch_b are answered using $\mathcal{B}$'s $\mathsf{SIGCHAL}_b$ oracle. This simulation is perfect. If $\mathcal{A}$ distinguishes with non-negligible advantage, this directly leads to a non-negligible advantage for $\mathcal{B}$ in UAS's signature anonymity game. ■

Theorem 9 (Traceability of $\Pi^{\text{gs}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction has signature unforgeability according to Definition 5, and issuance unforgeability according to Definition 4, then $\Pi^{\text{gs}}_{\mathsf{UAS}}$ (Section 6.2) is a traceable group signature scheme.*

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{Exp}^{\text{trace}}_{\mathcal{A},\text{gs}}$ for our $\Pi^{\text{gs}}_{\mathsf{UAS}}$ construction. Now, consider the following sequence of games:

G_0 This is the standard $\mathsf{Exp}^{\text{trace}}_{\mathcal{A},\text{gs}}$.

G_1 Behaves like G_0 but it computes par' as $(par', \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$ (note that the distribution of par' does not change). Furthermore:

- Whenever $\mathcal{A}$ queries $\mathsf{ISSUE}(upk)$ and the issuer does not output $\perp$, then G_1 computes $(upk', \cdot) \leftarrow \mathsf{ExtIss}(\tau, reg)$ using the UAS extractor ExtIss. If $upk \neq upk'$, i.e. if the extracted upk' is inconsistent to the desired upk according to the oracle query, then G_1 aborts and outputs 0 (making $\mathcal{A}$ lose). We call the event that this happens $\texttt{FailIss}$.

Clearly, there is no difference between G_0 and G_1 unless the $\texttt{FailIss}$ error event happens, hence $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| \leq \Pr[\texttt{FailIss}]$ (via difference lemma).

Through a straightforward reduction to UAS's issuance unforgeability (in which $\mathcal{B}$ guesses the oracle query that triggers $\texttt{FailIss}$), one can show that $\Pr[\texttt{FailIss}]$ is negligible. This is because $\texttt{FailIss}$ effectively triggers the win condition "$f_{\text{is}}^{\text{upk}}(upk', \dots) \neq y_{\text{is}} = upk$" in line 13 of $\mathsf{Exp}^{\text{iss-forge}}_{\mathsf{UAS},\mathcal{A}}$.

Now, consider $\mathcal{A}$ playing against G_1. We can build an adversary $\mathcal{B}$ that breaks signature unforgeability of the underlying Π_{UAS} construction. The simulation of $\mathcal{A}$'s environment is again direct, setting up honest issuer iid for $f_{\text{is}}^{\text{upk}}$ and honest opener oid. Eventually, $\mathcal{A}$ outputs a (m, σ) pair.

When that happens, $\mathcal{B}$ outputs $(\mathsf{oid}, \mathsf{iid}, \Sigma = (\sigma, 0), m, f_{\text{ev}}^0)$ in its signature unforgeability game. Note that $\mathcal{B}$ has not requested any UAS signatures from its oracles.

If $\mathcal{A}$ wins G_1 because Judge outputs 0, then this clearly translates to a win for $\mathcal{B}$ in line 17/18 of $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$. If $\mathcal{A}$ wins G_1 because $\mathcal{A}$ made no queries to $\mathsf{ISSUE}(upk)$, then $\mathcal{B}$ has not queried its own $\mathsf{ISSUE}(\dots, y_{\text{is}} = upk)$ oracle. Hence $\mathcal{B}$ wins in line 9 of $\mathsf{CheckEndorsementCreds}$, given that, because of $\neg\texttt{FailIss}$, the extractor $ExtractIssue$ only ever outputs upk' that have been queried to $\mathsf{ISSUE}(\dots, y_{\text{is}} = upk')$ (but upk has not been queried to ISSUE). This implies that $\Pr[\mathsf{G}_1 = 1]$ must be negligible. Hence $\Pr[\mathsf{G}_0 = 1]$ is negligible, as required. ■

Theorem 10 (Non-frameability of $\Pi^{\text{gs}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction is non-frameable according to Definition 6, then $\Pi^{\text{gs}}_{\mathsf{UAS}}$ (Section 6.2) is a non-frameable group signature scheme.*

Proof. Assume an adversary $\mathcal{A}$ against non-frameability in $\Pi^{\text{gs}}_{\mathsf{UAS}}$. We build an adversary $\mathcal{B}$ against non-frameability of the $(f_{\text{is}}^{upk}, f_{\text{ev}}^0, f_{\text{op}}^{upk}) - \Pi_{\mathsf{UAS}}$ instance.

$\mathcal{B}$'s simulation of $\mathcal{A}$'s environment is as in the proof for traceability of $\Pi^{\text{gs}}_{\mathsf{UAS}}$, except that now $\mathcal{B}$ also corrupts the issuer by running ICORR, so that it can pass isk to $\mathcal{A}$. The simulation is again perfect.

$\mathcal{B}$ takes $\mathcal{A}$'s (m, σ, upk, π) output, and augments it to $(\mathsf{oid}, \mathsf{iid}, (\sigma, 0), m, f_{\text{ev}}^0, upk, \pi)$, where oid and iid are the identifiers of the produced opener and issuer, respectively. Note that, if $\mathcal{A}$ wins in its GS non-frameability game, then $\mathcal{B}$ also directly wins in its UAS non-frameability game. Concretely, if $\mathcal{A}$'s signature is accepted by Verify in the $\Pi^{\text{gs}}_{\mathsf{UAS}}$ construction, then by definition it is also accepted by Π_{UAS} Verify algorithm. The same applies if (upk, π) is accepted by $\Pi^{\text{gs}}_{\mathsf{UAS}}$ Judge. In addition, since $\mathcal{A}$ wins, then it has not leveraged the SIGN oracle to produce σ, and thus $\Sigma = (\sigma, 0) \notin \mathsf{SIG}$. Finally, if $upk \in \mathsf{HU}$ and $\mathcal{A}$ did not query UCORR on upk, then $\mathcal{B}$ did not query UCORR on uid such that $\mathsf{UPK}[\mathsf{uid}] = upk$, so $\mathcal{B}$ wins its non-frameability game at line 12 or 14. ■

D.3 Anonymous Credentials

We adopt the model in [FHS19] for anonymous credentials with selective disclosure, which restricts to one credential per presentation. Therein, an anonymous credential scheme is defined via OrgKeyGen (which we rename to IssKeyGen), and UserKeyGen algorithms, and $\langle \text{Obtain}, \text{Issue} \rangle$

and $\langle \text{Show}, \text{Verify} \rangle$ interactive protocols. We refer to [FHS19] for the full details. For readability, we give the security definitions in Fig. 20.

$\mathsf{Exp}^{\text{anon-}b}_{\mathcal{A},\text{ac}}(1^\kappa)$

$(ipk, isk) \leftarrow \text{IssKeyGen}(1^\kappa)$
$b^* \leftarrow \mathcal{A}^{\mathsf{UGEN},\mathsf{UCORR},\mathsf{OBTAIN},\mathsf{Show},\mathsf{LoR}}(ipk, isk)$
return b^*

$\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{ac}}(1^\kappa)$

$(ipk, isk) \leftarrow \text{IssKeyGen}(1^\kappa)$
$(D, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{UGEN},\mathsf{UCORR},\mathsf{OBTISS},\mathsf{ISSUE},\mathsf{Show}}(ipk)$
Let $\boldsymbol{d}$ be the index set of D, i.e. $D \in \mathcal{AS}^{\boldsymbol{d}}$
$\langle \cdot, b \rangle \leftarrow \langle \mathcal{A}(\mathsf{st}), \text{Verify}(ipk, \boldsymbol{d}, D) \rangle$
return $(b = 1 \wedge \nexists j \ s.t.$
$(\mathsf{OWN}[j] \notin \mathsf{HU} \wedge D \subseteq \mathsf{ATT}[j]))$

Figure 20: Games for anonymous credentials with selective disclosure [FHS19]. OWN, ATT and the oracles are essentially as in UAS, but Show runs the (challenge-response) Show protocol of the AC construction with $\mathcal{A}$ as the verifier. $\mathcal{A}$ wins $\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{ac}}$ if it authenticates successfully while revealing attributes not contained in any adversarially controlled credential.

Security of $\Pi^{\text{ac}}_{\mathsf{UAS}}$ We prove that $\Pi^{\text{ac}}_{\mathsf{UAS}}$ (Section 6.3) is an anonymous and unforgeable anonymous credential scheme, according to [FHS19], if the underlying Π_{UAS} construction is anonymous and unforgeable.

Theorem 11 (Anonymity of $\Pi^{\text{ac}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction has signature anonymity according to Definition 3, then $\Pi^{\text{ac}}_{\mathsf{UAS}}$ (Section 6.3) is an anonymous AC scheme according to [FHS19].*

Proof. Given $\mathcal{A}$ against anonymity of $\Pi^{\text{ac}}_{\mathsf{UAS}}$ as defined in [FHS19], we build an adversary $\mathcal{B}$ against signature anonymity of Π_{UAS} as defined in Definition 3.

$\mathcal{B}$ prepares $\mathcal{A}$'s game by running HUGEN and ISET (specifying $f^{\mathbf{upk}}_{\text{is}}$ as issuance function) to produce an (isk, ipk) issuer key pair, and then runs ICORR to learn isk, which passes to $\mathcal{A}$. To simulate $\mathcal{A}$'s oracle calls, $\mathcal{B}$ leverages its own oracles in the UAS signature anonymity game. Show calls are simulated by SIGN (setting m to $\mathcal{A}$'s challenge, and f_{ev} to $f^{\boldsymbol{d}}_{\text{ev}}$), and LoR calls are simulated with $\mathsf{SIGCHAL}_b$ (again, setting m to $\mathcal{A}$'s challenge, and using $f^{\boldsymbol{d}}_{\text{ev}}$). Note that, in the latter, if either of the challenge credentials does not contain the necessary attribute set (defined by $f^{\boldsymbol{d}}_{\text{ev}}$), $\mathsf{SIGCHAL}_b$ aborts (as LoR does). The simulation is perfect.

$\mathcal{B}$ simply outputs whatever $\mathcal{A}$ does. Clearly, if $\mathcal{A}$ has non-negligible distinguishing advantage, then so does $\mathcal{B}$ in the UAS signature anonymity game, with the same advantage. ■

Theorem 12 (Unforgeability of $\Pi^{\text{ac}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction has signature unforgeability according to Definition 5, then $\Pi^{\text{ac}}_{\mathsf{UAS}}$ (Section 6.3) is an unforgeable AC scheme.*

We prove the theorem by direct reduction to the signature unforgeability property. Intuitively, an adversary who convinces an honest $\Pi^{\text{ac}}_{\mathsf{UAS}}$ verifier has to forge an UAS signature.

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{ac}}(1^\kappa)$. We construct $\mathcal{B}$ against UAS signature unforgeability. $\mathcal{B}$ gets par' as input. $\mathcal{B}$ uses its $\mathsf{UGEN}, \mathsf{ISET}$ oracles to set up an honest issuer with ID iid and key ipk' for $f^{\mathbf{upk}}_{\text{is}}$. It further uses HOGEN to set up an honest opener with ID oid for f^0_{op}. It then sets $par = (par', opk)$ and $ipk = (ipk', f^{\mathbf{upk}}_{\text{is}})$ and hands par, ipk to $\mathcal{A}$.

$\mathcal{B}$ then answers $\mathcal{A}$'s oracle queries using its own UAS oracles in the natural way. The simulation is perfect (note that the par' that $\mathcal{A}$ sees when run as a subroutine by $\mathcal{B}$ are generated by $\mathsf{SimSetup}$ rather than Setup, but by definition, this does not change the distribution of par' at all).

When $\mathcal{A}$ returns D and st, our adversary $\mathcal{B}$ runs $\mathcal{A}(\mathsf{st})$ and sends a random nonce $r \xleftarrow{\$} \{0,1\}^\kappa$ to $\mathcal{A}$. $\mathcal{A}$ responds with a signature σ. $\mathcal{B}$ outputs $(\mathsf{oid}, \mathsf{iid}, \Sigma = (\sigma, y_{\text{ev}} = D), m = r, f_{\text{ev}}^{\boldsymbol{d}})$ (where $\boldsymbol{d}$ is the adequate index set for D) as a UAS signature forgery.

Let $\texttt{coll}$ be the event that the value random r has been queried before to SIGN. Because r is chosen uniformly at random from a size 2^κ set, this happens only with negligible probability.

We now argue that whenever $\mathcal{A}$ outputs a valid AC forgery and $\neg\texttt{coll}$, then $\mathcal{B}$ outputs a valid UAS signature forgery. Because $\neg\texttt{coll}$, the forgery of $\mathcal{B}$ survives the check in line 6 of $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ (Figure 12). If $\mathcal{A}$ wins its game, then the signature is valid, hence $\mathcal{B}$'s forgery also survives the check in line 8. Finally, there are two possible cases, in both of which $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ outputs 1:

- If $f_{\text{ev}}^{\boldsymbol{d}}(\cdot,(\cdot,\boldsymbol{a})) \neq D$ for the extracted $\boldsymbol{a}$, then $\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}$ outputs 1 in line 1
- If $f_{\text{ev}}^{\boldsymbol{d}}(\cdot,(\cdot,\boldsymbol{a})) = D$, then $\mathsf{CheckEndorsementCreds}$ necessarily outputs 1 in line 3 or in line 5 as guaranteed by the AC unforgeability winning condition (stating that the honest issuer has not issued any credential with fitting attributes to a corrupted party).

Overall, we get that $\Pr[\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{ac}}(1^\kappa) = 1] \leq \Pr[\mathsf{Exp}^{\text{sig-forge}}_{\mathsf{UAS},\mathcal{A}}(1^\kappa) = 1] + \Pr[\texttt{coll}]$, which is negligible by assumption. ■

Note that the security definition [FHS19] does not require credentials to be bound to identities, hence there is no need for an honest issuer to actually check the issuance utility $y_{\text{is}} = upk$ value (it does not appear in the security proof). However, our UAS definitions are much more strict w.r.t. credential ownership, i.e. $\Pi^{\text{ac}}_{\mathsf{UAS}}$ actually fulfills a stronger security definition that takes credential owners into account. Such a non-transferability winning condition would correspond to line 9 of $\mathsf{CheckEndorsementCreds}$.

D.4 Ring Signatures

We adopt the model in [BKM06], where security of ring signatures is defined according to the games in Fig. 21. More specifically, we give next their formulation of anonymity against full key exposure, and unforgeability with respect to insider corruption (the stronger model among the variants given in [BKM06]). We refer to [BKM06] for details.

Security of $\Pi^{\text{ring}}_{\mathsf{UAS}}$ We prove that our $\Pi^{\text{ring}}_{\mathsf{UAS}}$ construction (Section 6.4) is anonymous and unforgeable, in the model given in [BKM06], if the underlying Π_{UAS} construction is secure.

Theorem 13 (Anonymity of $\Pi^{\text{ring}}_{\mathsf{UAS}}$)**.** *If the base Π_{UAS} construction is anonymous according to Definition 3, then $\Pi^{\text{ring}}_{\mathsf{UAS}}$ (Section 6.4) is an anonymous ring signature scheme.*

Proof. Assume $\mathcal{A}$ is an adversary against anonymity of $\Pi^{\text{ring}}_{\mathsf{UAS}}$. We build $\mathcal{B}$ that breaks signature anonymity of the underlying Π_{UAS} construction.

To simulate $\mathcal{A}$'s environment, when $\mathcal{B}$ receives the system parameters in the UAS anonymity game, it runs the HOGEN oracle with f^0_{op}, to get opk. Then, it generates n honest users by running the HUGEN oracle as many times. Finally, passes the n user public keys to $\mathcal{A}$. To simulate $\mathcal{A}$'s

$\mathsf{Exp}^{\text{anon-}b}_{\mathcal{A},\text{rs}}(1^\kappa)$

1 : $par \leftarrow \text{Setup}(1^\kappa)$
2 : **for** $i \in [n] : (pk_i, sk_i) \leftarrow \text{KG}(par)$
3 : $(i_0, i_1, \boldsymbol{ring}, m, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{Sign}}(pk_1, \ldots, pk_n)$
4 : **if** $\{pk_{i_0}, pk_{i_1}\} \not\subseteq \boldsymbol{ring} \vee i_0 = i_1 :$ **return** $\perp$
5 : $\sigma^* \leftarrow \text{Sign}(i_b, \boldsymbol{ring}, m)$
6 : $b^* \leftarrow \mathcal{A}(\sigma^*, \{sk_i\}_{i\in[n]}, \mathsf{st})$
7 : **return** b^*

$\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{rs}}(1^\kappa)$

1 : $par \leftarrow \text{Setup}(1^\kappa)$
2 : **for** $i \in [n] : (pk_i, sk_i) \leftarrow \text{KG}(par)$
3 : $(\boldsymbol{ring}, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign},\mathsf{Corr}}(pk_1, \ldots, pk_n)$
4 : **if** $\text{Verify}(\boldsymbol{ring}, \sigma, m) = 0 :$ **return** 0
5 : **return** $(\boldsymbol{ring} \subseteq \{pk_i\}_{i\in[n]} \setminus CU \wedge$
6 : $\quad \mathcal{A}$ never queried $\mathsf{Sign}(\cdot, \boldsymbol{ring}, m))$

Figure 21: Security games for ring signatures [BKM06]. The Sign oracle accepts (i, R, m) tuples and, if $pk_i \in R$, adds the tuple to a list of queries, and returns $\sigma \leftarrow \text{Sign}(sk_i, R, m)$. Corr accepts an index i, and leaks sk_i. Corrupted users are added to CU.

queries to the Sign oracle, $\mathcal{B}$ leverages its own SIGN oracle. Namely, when receiving a $\mathsf{Sign}(i, R, m)$ call, $\mathcal{B}$ checks that $\mathsf{UPK}[i] \in R$ and, if affirmative, precomputes f^R_{ev} and calls SIGN on $(\mathsf{oid}, i, \emptyset, m, f^R_{\text{ev}})$.

Eventually, $\mathcal{A}$ outputs $(i_0, i_1, \boldsymbol{ring}, m, \mathsf{st})$. If $i_0 \neq i_1$, and both $\mathsf{UPK}[i_0], \mathsf{UPK}[i_1]$ belong to $\boldsymbol{ring}$, then $\mathcal{B}$ calls its $\mathsf{SIGCHAL}_b$ oracle on $(i_0, i_1, \emptyset, m, f^{\boldsymbol{ring}}_{\text{ev}})$ to get a signature Σ^* (otherwise, it aborts). $\mathcal{B}$ calls UCORR on all the honest users previously generated (including the ones for i_0 and i_1), and passes all secret keys along with Σ^* and st to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ does. Note that the simulation, for which $\mathcal{B}$ simply leverages its own oracles, is perfect. Clearly, if $\mathcal{A}$ has non-negligible advantage in its RS anonymity game, then so does $\mathcal{B}$, with the same advantage, in its UAS signature anonymity game. ■

Theorem 14 (Unforgeability of $\Pi^{\text{ring}}_{\mathsf{UAS}}$)**.** *If the base* Π_{UAS} *construction is non-frameable according to Definition 6, and has signature unforgeability as defined in Definition 5, then* $\Pi^{\text{ring}}_{\mathsf{UAS}}$ *(Section 6.4) is an unforgeable ring signature scheme.*

Similar to the proof of the digital signature construction (Theorem 7), we need to handle the event `FailEv` that the extractor outputs values for which $f_{\text{ev}}(\cdot) \neq 1$, and the event `FailOp` that we cannot honestly compute an opening proof for the forgery. Both events are unlikely because of the Π_{UAS} signature unforgeability property. As soon as this is done, we can reduce to the Π_{UAS} traceability property, which implies that it's hard to create new signatures that the extractor assigns to an honest user. Given $\neg$`FailEv`, we know that the extractor assigns the signature to one of the users in the ring, which concluces the proof.

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{rs}}(1^\kappa)$ for our $\Pi^{\text{ring}}_{\mathsf{UAS}}$ construction. Now consider the following sequence of games:

G_0 This is the standard $\mathsf{Exp}^{\text{forge}}_{\mathcal{A},\text{rs}}(1^\kappa)$.

G_1 Behaves like G_0 but it computes par' as $(par', \tau) \leftarrow \mathsf{SimSetup}(1^\kappa)$ (note that the distribution of par' does not change). Furthermore, when $\mathcal{A}$ outputs a winning forgery $(\boldsymbol{ring}, m, \sigma)$:

- G_1 runs the UAS extractor $(upk', \cdot) \leftarrow \mathsf{ExtSign}(\tau, \Sigma)$ with the appropriate $\Sigma = (\sigma, 1)$ (where $y_{\text{ev}} = 1$ is the membership bit output by f_{ev}). If the extracted upk' is not part of $\boldsymbol{ring}$, then G_1 outputs 0. We call the event that this happens `FailEv`.

- G_1 computes an opening proof $(y_{\text{op}}, \pi_{\text{op}}) \leftarrow \text{Open}(osk, \emptyset, \Sigma, m, f_{\text{ev}}^{\text{upk}})$ for the signature output by $\mathcal{A}$. If π_{op} is invalid, i.e. $y_{\text{op}} \neq 0$ or $\text{Judge}(opk, \emptyset, y_{\text{op}}, \pi_{\text{op}}, \Sigma, m, f_{\text{ev}}^{\text{upk}}) = 0$ (where osk is as computed during $\Pi_{\mathsf{UAS}}^{\text{ds}}$.Setup), then G_1 outputs 0. We call the event that this happens `FailOp`.

Clearly, there is no difference between G_0 and G_1 unless one of the two error events happens, hence $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| \leq \Pr[\texttt{FailEv} \vee \texttt{FailOp}]$ (via difference lemma).

We argue that the signature unforgeability property of UAS ensures that $\Pr[\texttt{FailEv} \vee \texttt{FailOp}]$ is negligible. In the following, we consider an adversary $\mathcal{B}$ against $\mathsf{Exp}_{\mathsf{UAS},\mathcal{B}}^{\text{sig-forge}}$, leveraging $\mathcal{A}$ against $\mathsf{Exp}_{\mathcal{A},\text{rs}}^{\text{forge}}(1^\kappa)$. $\mathcal{B}$ uses its oracles to set up n honest users. Let uid_i, upk_i be those honest users' IDs and public keys, respectively. $\mathcal{B}$ uses those users' upk_i when simulating $\mathsf{Exp}_{\mathcal{A},\text{rs}}^{\text{forge}}$ for $\mathcal{A}$. It also sets up an honest opener for $f_{\text{op}} = f_{\text{op}}^0$ and subsequently uses its corresponding opk for the simulation. Let oid be that opener's ID. Whenever $\mathcal{A}$ asks for a signature, $\mathcal{B}$ uses its own UAS SIGN oracle in the natural way to answer. Whenever $\mathcal{A}$ asks for corruption, $\mathcal{B}$ uses its own UAS corruption oracle UCORR to answer. When $\mathcal{A}$ outputs its candidate forgery $(\boldsymbol{ring}, m, \sigma)$ and wins its unforgeability game, then $\mathcal{B}$ outputs $(\mathsf{oid}, \emptyset, \Sigma = (\sigma, 1), m, f_{\text{ev}}^{\boldsymbol{ring}})$ to $\mathsf{Exp}_{\mathsf{UAS},\mathcal{A}}^{\text{sig-forge}}$. It is easy to see that `FailEv` and `FailOp` both correspond to winning cases for $\mathcal{B}$: If `FailEv` happens, then $\mathcal{B}$ wins in line 12 of $\mathsf{Exp}_{\mathsf{UAS},\mathcal{A}}^{\text{sig-forge}}$, if `FailOp` happens, then $\mathcal{B}$ wins in line 17/18. Because $\mathcal{B}$ is a ppt and Π_{UAS} is unforgeable, $\Pr[\texttt{FailEv} \vee \texttt{FailOp}]$ must be negligible (and hence G_0 and G_1 are indistinguishable).

Now consider $\mathcal{A}$ playing against G_1. We can build an adversary $\mathcal{B}$ that breaks non-frameability of the underlying Π_{UAS} construction. The simulation of $\mathcal{A}$'s environment is again direct. Eventually, $\mathcal{A}$ outputs a (m, σ) pair.

When that happens, $\mathcal{B}$ outputs $(\mathsf{oid}, \emptyset, \Sigma = (\sigma, 1), m, f_{\text{ev}}^{\boldsymbol{ring}}, 0, \pi_{\text{op}})$ in its non-frameability game. If $\mathcal{A}$ wins G_1, then σ does not belong to SIG, and hence it is accepted by Verify in $\mathcal{B}$'s non-frameability game. Moreover, Judge accepts π_{op} (because $\neg\texttt{FailOp}$ is required for $\mathcal{A}$ to win G_1). Moreover, because of $\neg\texttt{FailEv}$, we also know that upk output by $\mathsf{ExtSign}$ upk satisfies $upk \in \boldsymbol{ring}$ and hence $upk \notin \mathsf{CU}$. Thus, whenever $\mathcal{A}$ wins G_1, then $\mathcal{B}$ wins its non-frameability game in line 14. This implies that $\Pr[\mathsf{G}_1 = 1]$ must be negligible. Hence $\Pr[\mathsf{G}_0 = 1]$ is negligible, as required. ■

E Relationships with More Schemes

Next, we give the intuition on how to build other known and more advanced schemes – or variants of them – and specify the corresponding Π_{UAS} restrictions.

E.1 Group Signatures with Message Dependent Opening

We describe a direct approach to build a GS scheme in which only signatures over pre-agreed messages can be deanonymized. This resembles the work on group signatures with message dependent opening (GS-MDO) described in [EHK+19]. The GS-MDO setting features two separate authorities: the opener, who opens signatures; and the admitter, who computes per-message tokens to grant the opener permission to open signatures over the corresponding concrete messages. Without these per-message tokens generated by the admitter, the opener cannot open signatures.

We must however observe that we cannot build the exact same functionality, nor follow the same model, from UAS. Firstly because of the authority separation aspect at the opening level. Secondly,

because in GS-MDO, the admitter can enable the opener to open a signature *after* the user has produced it – i.e., GS-MDO has fully authority-controlled opening, even despite the separation.

Concerning the first aspect, it is easy to see that our UAS model does not include authority separation at the opening level. Anyway, this can be somehow mimicked via conventional techniques (like applying threshold cryptography to the authorities' keys), which we leave out of scope for the sake of simplicity. On the other hand, the authority-controlled opening aspect is in some sense philosophically differentiating. In GS-MDO, when a user produces a signature, she cannot know whether it will be openable in the future – and this is independent from the authority separation aspect. For instance, a rogue admitter might decide to always grant the opener "full opening" capabilities. In that case, GS-MDO collapses to a conventional GS scheme. As opposed to this approach, in our Π_{UAS} construction, the user always knows at signature generation time what information an opener may be able to extract in the future. A related notion is the *user-controlled linkability* from [DL21]. In that sense, one can see our variant of GS-MDO as GS-MDO with user-controlled linkability, or GS-MDO-UCL.

Thus, in a nutshell, what GS-MDO provides is a scheme that allows users to produce anonymous signatures over arbitrary messages, which openers can deanonymize at any point in time if the admitter decides that the signed message should be deanonymizable. In contrast, what we describe next is how to build, from Π_{UAS}, a scheme that allows users to produce anonymous signatures over arbitrary messages, which openers can deanonymize *only if* the signed message was marked as deanonymizable beforehand. While different as argued above, we still believe that the "message-dependent opening" aspect is common to both.

Building $\Pi_{\mathsf{UAS}}^{\mathrm{gs-mdo-ucl}}$. The approach to build this variant of GS-MDO from Π_{UAS} is simple: given $f_{\mathrm{op}}^{\boldsymbol{msg}}$ as defined in Equation (1), and f_{is}^{upk} and f_{ev}^{0} as defined in Section 6, our $\Pi_{\mathsf{UAS}}^{\mathrm{gs-mdo-ucl}}$ construction is simply a $(f_{\mathrm{is}}^{upk}, f_{\mathrm{ev}}^{0}, f_{\mathrm{op}}^{\boldsymbol{msg}})$-$\Pi_{\mathsf{UAS}}$ restriction. It is easy to see that this provides the functionality hinted above: any user can get at most one membership credential, no information is revealed alongside the (group) signatures, and the opener can only deanonymize the signatures over messages in $\boldsymbol{msg}$. Note that, while the opener has the capability to define a new function over a new $\boldsymbol{msg}$ set, the signer always knows what messages are part of $\boldsymbol{msg}$ *at signature generation time*. Thus, the signer knows if a signature she produces will ever be deanonymizable or not.

$$f_{\mathrm{op}}^{\boldsymbol{msg}}(upk, \cdot, m) \coloneqq \textbf{if } m \in \boldsymbol{msg} : \textbf{return } upk; \textbf{ else return } 0 \tag{1}$$

E.2 Multimodal Private Signatures

Multimodal Private Signatures [NGSY22b] (or, rather, a close relative, see next) can also be implemented as a Π_{UAS} restriction.

The idea behind MPS is that the signed message determines the level to which the user's identity id is disclosed to the opener. The function $F(m, w, id)$ determines the level of disclosure (and whether that user is allowed to sign that message m at all). The functions $G_1, \ldots, G_n$ then model what information is disclosed for each level. Overall, the opener learns $G_{F(m,w,id)}(id)$ about the user's identity, and $F(\cdot) = 0$ signifies that the user is not allowed to sign the message at all.

For our instantiation, we replace the user's ID id with the user's upk. Furthermore, we omit the witness w from F, and instead let F use attributes $\boldsymbol{a}$ from the user's credential/certificate. This is a slightly weaker notion, as w in [NGSY22b] could be chosen depending on the current message (e.g.,

m could be a commitment and w its opening value). However, we believe that this is a reasonable simplification. To recover the original notion, we can let the user self-issue a credential containing the attribute w, at which point F within $f_{\text{ev}}, f_{\text{op}}$ can depend on it.

Building $\Pi^{\text{mps}}_{\mathsf{UAS}}$. For simplicity, we only give the functions needed to represent the $\Pi^{\text{mps}}_{\mathsf{UAS}}$ variant in which w is assumed to be of the $(cid, \boldsymbol{a})$ form. The generalization to more flexible w values would need to take into account all the used credentials (and their attributes) in the evaluation and opening functions, following the intuition given above.

In a nutshell, $\Pi^{\text{mps}}_{\mathsf{UAS}}$ can be built as a $(f^{upk}_{\text{is}}, f^{F}_{\text{ev}}, f^{G}_{\text{op}})$-$\Pi_{\mathsf{UAS}}$ restriction, where f^{upk}_{is} is as defined in Section 6, and f^{F}_{ev} and f^{G}_{op} are as defined in Equations (2). Note that f^{F}_{ev} outputs 1 if F does not output 0, and 0 otherwise, meaning that the signer is not allowed to sign this message at all. f^{G}_{op} is simply MPS's G function.

$$
\begin{aligned}
&f^{F}_{\text{ev}}(upk, (cid, \boldsymbol{a}), m) \coloneqq \\
&\qquad \textbf{if } F(upk, (cid, \boldsymbol{a}), m) = 0 : \textbf{return } 0 \\
&\qquad \textbf{else } : \textbf{return } 1 \\
&\quad f^{G}_{\text{op}}(upk, cid, \boldsymbol{a}, \cdot) \coloneqq \\
&\qquad \textbf{return } G_{F(upk,(cid,\boldsymbol{a}),m)}((upk, cid, \boldsymbol{a}))
\end{aligned} \tag{2}
$$

E.3 Revocable Anonymous Credentials

From Π_{UAS}, it is also possible to build the approach to revocable anonymous credentials based on lists of revoked credentials. Therein, some authority maintains a list of the credentials that should not (or should, in the whitelisting variant) be allowed to produce signatures [CKL+15, BCD+17, WG23]. Then, in order to produce a signature, a user has to prove knowledge of a credential whose identifier has not been added to that revocation list (resp. has been added, in the whitelisting variant). Since privacy needs to be ensured, this proof is done using zero-knowledge techniques. Note also that, since UAS has opening, our approach to revocable credentials supports *signature-driven* revocation straight away, as opposed to (the perhaps more usual in the academia) issuer-driven revocation. In practice, this is useful, as verifiers (service providers) can directly provide the evidence (signature) of a misbehavior, and have the associated credential revoked. This is not possible with issuer-driven revocation, where only issuers can revoke credentials (e.g., when a user warns about a lost credential).

Building $\Pi^{\text{rac}}_{\mathsf{UAS}}$. Briefly, this is achieved via a $(f^{\mathtt{upk}}_{\text{is}}, f^{\mathtt{cid},\boldsymbol{d},L}_{\text{ev}}, f^{\mathtt{cid}}_{\text{op}})$-$\Pi_{\mathsf{UAS}}$ restriction, where $f^{\mathtt{upk}}_{\text{is}}$ is as in Section 6, and $f^{\mathtt{cid},\boldsymbol{d},L}_{\text{ev}}$ and $f^{\mathtt{cid}}_{\text{op}}$ are as specified in Equations (3).

$$
\begin{aligned}
&f^{\mathtt{cid},\boldsymbol{d},L}_{\text{ev}}(upk, (cid, \boldsymbol{a}), \cdot) \coloneqq \\
&\qquad \textbf{if } cid \in L : \textbf{return } 0 \\
&\qquad \textbf{else return } \{\boldsymbol{a}_i\}_{i \in \boldsymbol{d}} \\
&\quad f^{\mathtt{cid}}_{\text{op}}(\cdot, (cid, \boldsymbol{a}), \cdot) \coloneqq \textbf{return } cid
\end{aligned} \tag{3}
$$

Concretely, $f_{\text{ev}}^{\texttt{cid},\boldsymbol{d},L}$ is defined via a set of attributes $\boldsymbol{d}$ to reveal (which must include the credential identifier *cid*), and a list L of revoked credentials. Note that these parameters can be hardcoded, though: the indexes of $\boldsymbol{d}$ are also fixed (e.g., per credential type), and L can be updated per verifier with some reasonable frequency – e.g., daily (of course, revocation of a credential is not effective until L, and the function, is updated). If *cid* appears in L, then the credential used for signing has been revoked, and $f_{\text{ev}}^{\texttt{cid},\boldsymbol{d},L}$ outputs 0, indicating that the signature must be rejected. Otherwise, $f_{\text{ev}}^{\texttt{cid},\boldsymbol{d},L}$ reveals the requested attributes by setting $y_{\text{ev}} \leftarrow \{\boldsymbol{a}_i\}_{i\in\boldsymbol{d}}$.

If a verifier wants to revoke a credential, all it has to do is ask the opener to run Open over a received signature produced from the credential to be revoked. This outputs *cid*, along with a proof π of correct opening. The verifier checks the proof and, if correct, appends *cid* to L.