Practical Group-Signatures with Privacy-Friendly Openings

Stephan Krenn[1], Kai Samelin[2], and Christoph Striecks[1]

[1] AIT Austrian Institute of Technology, Vienna, Austria
{stephan.krenn,christoph.striecks}@ait.ac.at
[2] TÜV Rheinland i-sec GmbH, Hallbergmoos, Germany
kaispapers@gmail.com

Abstract. Group signatures allow creating signatures on behalf of a group, while remaining anonymous. To prevent misuse, there exists a designated entity, named the opener, which can revoke anonymity by generating a proof which links a signature to its creator. Still, many intermediate cases have been discussed in the literature, where not the full power of the opener is required, or the users themselves require the power to claim (or deny) authorship of a signature and (un-)link signatures in a controlled way. However, these concepts were only considered in isolation.
We unify these approaches, supporting all these possibilities simultaneously, providing fine-granular openings, even by members. Namely, a member can prove itself whether it has created a given signature (or not), and can create a proof which makes two created signatures linkable (or unlinkable resp.) in a controlled way. Likewise, the opener can show that a signature was not created by a specific member and can prove whether two signatures stem from the same signer (or not) without revealing anything else. Combined, these possibilities can make full openings irrelevant in many use-cases. This has the additional benefit that the requirements on the reachability of the opener are lessened. Moreover, even in the case of an involved opener, our framework is less privacy-invasive, as the opener no longer requires access to the signed message.
Our provably secure black-box CCA-anonymous construction with dynamic joins requires only standard building blocks. We prove its practicality by providing a performance evaluation of a concrete instantiation, and show that our non-optimized implementation is competitive compared to other, less feature-rich, notions.

1 Introduction

Group signatures (Ω) became an integral tool for a lot of higher-level protocols, such as anonymous credentials. Originally introduced by Chaum and van Heyst [18], Ω schemes allow a member to sign arbitrary messages on behalf of the group without revealing its identity, while also prohibiting linking signatures. However, in case of a dispute, a dedicated third party (known as *inspector*, *(group) opener*, *judge*, or *tracing authority*) can later revoke the anonymity and make the signer accountable for the signature, i.e., *open* a signature.

This very basic definition of Ω schemes is, however, overly restricting in certain scenarios. For instance, as pointed out by Ishida et al. [24], in case of a dispute, the opener always has to reveal the identity of the signer, which is clearly privacy-invasive in situations where a simple yes/no information is sufficient, i.e., where it is only important to know whether a specific member signed the document or not. To tackle this, they introduced the notion of *deniable* Ω schemes, where the opener not only can prove that a member signed a document, but can also prove that a specific member was *not* the signer – without revealing the identity of the actual signer.

While this reduces the privacy impact of the opening process, the approach by Ishida et al. still requires the opener to be involved in every opening process. This means that the opener learns a lot of additional information about the group and also the messages under dispute (note here that in the standard definition of group signatures [6], the opener learns the message). Furthermore, the opener can hardly be implemented as an offline party and it is unrealistic to assume that the necessary key material is shared among multiple parties to avoid abuse. Hence, the opener must be seen as a single point of failure regarding the members' privacy in Ω systems. It is therefore no surprise that this capability was also considered in the setting where a member itself, i.e., without the opener, can prove (or deny resp.) authorship of a signature [1,12,22,39].

1.1 Motivation

Still, there are a few open questions. (1) To open a signature, the opener requires a message *and* a signature before it can pinpoint the creator. This, however, is very privacy-invasive, as the opener always learns the message. So, can we somehow reduce the privacy-impact of this possibility? (2) As widely used in anonymous credentials [15,30], selective linkability also has its merits. So, why not take back a step and also grant the opener the possibility to create a proof that two signatures were created by the same signer without revealing the signer's identity and vice versa? (3) Selective linkability, as discussed above, still requires that the opener is involved. So what about letting a member decide when two signatures created by it should become linkable, especially if the opener does not need to be involved, further lowering the requirements of the opener? (4) All of the above possibilities increase the trust in the Ω scheme, while also limiting the privacy-invading nature of the opener and the proofs generated. However, can these possibilities be combined into one *practical* scheme? (5) Is it possible that users, openers and issuers can re-use keys across multiple groups, further reducing computational overhead?

1.2 Contribution

We answer those question to the affirmative by presenting a framework which combines all of the mentioned privacy-enhancing features and possibilities. Namely, in our new framework, members can claim themselves (or deny resp.) authorship of a signature and can disclose a proof whether two signatures created by it are linkable, related to the linkability property of direct anonymous attestation [14]. This already gives the members a huge amount of freedom, while the opener is no longer required to be queried for such proofs. Still, there are cases where the opener needs to be contacted, e.g., if a member is not willing to, or simply not able to, cooperate. Thus, we grant the opener the possibility to generate less privacy-invading proofs, i.e., it can prove whether a signature was created by a certain member or not and can also create a proof whether two signatures stem from the same signer (or not) without revealing any identities. Finally, an additional benefit of our framework is that all keys can be re-used, i.e., a user can join different groups with the same key pair, while also the keys of the opener can be used for different issuers and vice versa.

We provide suitable security definitions, and a provably secure black-box construction of such a Ω scheme, including a concrete instantiation. Both constructions are based on the efficient encrypt-and-prove paradigm, but are enriched with ideas originating from the concept of anonymous credentials. In particular, our construction is based on non-interactive zero-knowledge proofs of knowledge, IND-CPA secure encryption schemes, unforgeable signature schemes, and scope-exclusive pseudonym systems. To show that the resulting schemes are efficient enough for use in practice, we have evaluated the concrete scheme in Java.

We stress that our "basic" setup does not consider revocation, but dynamic joins. This was done to keep the model readable. However, extending our model to also cover revocation can be achieved straightforwardly using, e.g., the results by Baldimtsi et al. [4]. Moreover, due to our extended capabilities, the adversary must not be able to corrupt parties, as by the very goal of our framework, we cannot achieve forward-secrecy [38].

We emphasize that our framework is not supposed to replace existing ones in general, but should rather be seen as an extension which makes sense to deploy in specific scenarios. This decision depends on the very concrete scenario, and must consider aspects such as the acceptable level of trust into the opener, the frequency of expected opening requests, or the potential impact of coercion. We stress that in forward-secure Ω schemes, a member has to store the used randomness in order to prove that it signed a document, and thus coercion can also happen based on past signatures. Our construction is stateless in this regard.

Additional Related Work As already mentioned, Ω schemes were introduced by Chaum and van Heyst [18] as a privacy-preserving tool. The first thorough formal treatment of this primitive was then given by Bellare et al. [6], which introduce a sound security model for static groups, i.e., groups which are fixed at setup once and for all. This has later been extended for the case of dynamic groups [8,26], which has recently been revisited by Bootle et al. [13], including the case of revocation of members [4,16].

Due to the interesting nature of this primitive, there are a lot of constructions in the literature, based on a plethora of assumptions, different construction paradigms and possibilities. Constructions proposed include, but are not limited to, [3,10,11,28,32,37]. Also directly related is the concept of "traceable signatures" [25,29], where the release of a trapdoor allows to open the signatures from a specific member, but not the others. A nice overview of Ω schemes has been presented by Manulis et al. [31].

The idea of "self-traceable" group signatures was already mentioned by Song [38]. Namely, she argues that the leakage of the secret key may allow to "self-trace" all generated signatures, which she avoids by introducing forward-secure Ω schemes. However, as already clarified, this can actually be lifted to allow for something useful, if proofs can selectively be generated [1,12,22,39].

Likewise, the idea of (selectively) linkable group-signatures has been discussed [11,23,36]. However, we stress that in the those schemes the "linking authority" is different from the opener, holding its own secret key, which is not the case in our framework.

2 Preliminaries

The main security parameter is denoted by $\lambda \in \mathbb{N}$. All algorithms implicitly take 1^λ as an additional input. We write $a \leftarrow A(x)$ if a is assigned the output of the deterministic algorithm A with input x. If an algorithm A is probabilistic, we use $(a;r) \xleftarrow{\$} A(x)$ to make the randomness r drawn internally explicit for further usage. The randomness r may be dropped if clear from the context. An algorithm is efficient, if it runs in probabilistic polynomial time (PPT) in the length of its input. For the remainder of this paper, all algorithms are PPT if not explicitly mentioned otherwise. Most algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. If S is a set, we write $a \xleftarrow{\$} S$ to denote that a is chosen uniformly at random from S. In the definitions, we speak of a general message space $\mathcal{M}$ to be as generic as possible. What $\mathcal{M}$ is concretely, is defined in the instantiations. A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible, if it vanishes faster than every inverse polynomial, i.e., $\forall k \in \mathbb{N}$, $\exists n_0 \in \mathbb{N}$ such that $\nu(n) \leq n^{-k}$, $\forall n > n_0$. With $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ we denote a random oracle [7].

Building Blocks. The construction is based on the following building blocks:

1. An IND-CPA secure encryption scheme $\Pi = \{\mathsf{PGen}_\Pi, \mathsf{KG}_\Pi, \mathsf{Enc}, \mathsf{Dec}, \mathsf{KVf}_\Pi\}$, where the last algorithm, on input a public/private key pair, outputs a bit whether or not the keys correspond to each other[3];
2. An UNF-CMA secure signature scheme $\Sigma = \{\mathsf{PGen}_\Sigma, \mathsf{KG}_\Sigma, \mathsf{Sig}_\Sigma, \mathsf{Vf}_\Sigma\}$;
3. A weakly simulation-sound extractable non-interactive zero-knowledge system $\Xi = \{\mathsf{PGen}_\Xi, \mathsf{Prv}_\Xi, \mathsf{Vf}_\Xi\}$ that can be transformed into a signature of knowledge[4], and
4. A scope-exclusive pseudonym system $\Theta = \{\mathsf{PGen}_\Theta, \mathsf{KG}_\Theta, \mathsf{Gen}_\Theta\}$. Such a system allows to generate collision-resistant pseudonyms in a deterministic way based on some string sc (the "scope"). If different scs are used, the resulting pseudonyms are unlinkable across different users.

We stress that combining an IND-CPA secure encryption scheme with a Ξ implies IND-CCA2.

As Θs are not standard, their definition is restated next. All other definitions are given in Appendix A.

Pseudonym Systems In the following, we present pseudonym-systems in the fashion required [15]. In a nutshell, users can be known under different pseudonyms to different verifiers, being mutually unlinkable.

Framework. We now present the framework for pseudonym systems, taken from Camenisch et al. [15].

Definition 1 (Pseudonym Systems). *A pseudonym system Θ consists of three algorithms $\{\mathsf{PGen}_\Theta, \mathsf{KG}_\Theta, \mathsf{Gen}_\Theta\}$ such that:*

PGen_Θ**.** *This algorithm outputs parameters:*

$$\mathsf{pp}_\Theta \xleftarrow{\$} \mathsf{PGen}_\Theta(\mathsf{pp}_{\mathsf{SYS}})$$

The public parameters are assumed to be input to all following algorithms.

KG_Θ**.** *A user generates his secret key:*

$$\mathsf{usk} \xleftarrow{\$} \mathsf{KG}_\Theta(\mathsf{pp}_\Theta)$$

Gen_Θ**.** *A pseudonym* nym *for a given user secret key and a* $\mathsf{sc} \in \{0,1\}^*$ *is computed deterministically as:*

$$\mathsf{nym} \leftarrow \mathsf{Gen}_\Theta(\mathsf{usk}, \mathsf{sc})$$

Correctness. Correctness only requires that none of the above algorithms halts in an error state, when queried on honestly generated inputs.

[3] For simplicity, we assume that at most one secret key per public key exists.

[4] Formally, we use one proof-system for each of the proof goals involved in the construction; however, as there is no risk of confusion, we do not make this distinction explicit in the following.

Security. Collision resistance guarantees that for each scope string, any two users will have different pseudonyms with overwhelming probability. Finally, unlinkability guarantees that users cannot be linked across scopes.

Definition 2 (Collision Resistance). *A pseudonym system is* collision resistant*, if for every PPT algorithm $\mathcal{A}$ there is a negligible function ν such that:*

$$\begin{aligned}\Pr[&\mathsf{nym}_0 = \mathsf{nym}_1 \ \wedge\ \mathsf{usk}_0 \neq \mathsf{usk}_1 \ \wedge\ \mathsf{nym}_0 \neq \perp : \\ &\mathsf{nym}_0 \leftarrow \mathsf{Gen}_\Theta(\mathsf{usk}_0, \mathsf{sc}), \mathsf{nym}_1 \leftarrow \mathsf{Gen}_\Theta(\mathsf{usk}_1, \mathsf{sc}), \\ &(\mathsf{usk}_0, \mathsf{usk}_1, \mathsf{sc}) \xleftarrow{\$} \mathcal{A}(\mathsf{PGen}_\Theta(\mathsf{pp}_{\mathsf{SYS}}))] \leq \nu(\lambda)\end{aligned}$$

Definition 3 (Pseudonym Unlinkability). *We define pseudonym unlinkability as a game between the adversary and two oracles $\mathcal{O}_0(\mathsf{usk}_0, \cdot), \mathcal{O}_1(\mathsf{usk}_1, \cdot)$ simulating honest users as follows, where the oracles share an initially empty list L.*

We now require that for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have that for some negligible function ν it holds that:

$$\Pr[\mathsf{LinkNyms}^{\Theta}_{\mathcal{A}}(1^\lambda) = 1] \leq \tfrac{1}{2} + \nu(\lambda)$$

The corresponding experiment is depicted in Fig. 1.

```
**Experiment** $\mathsf{LinkNyms}^{\Theta}_{\mathcal{A}}(\lambda)$:
  $\mathsf{pp}_{\mathsf{SYS}} \xleftarrow{\$} \mathsf{PGen}_{\mathsf{SYS}}(1^\lambda)$
  $\mathsf{pp}_\Theta \xleftarrow{\$} \mathsf{PGen}_\Theta(\mathsf{pp}_{\mathsf{SYS}})$
  $\mathsf{usk}_i \xleftarrow{\$} \mathsf{KG}_\Theta(\mathsf{pp}_\Theta)$ for $i \in \{0, 1\}$
  $b \xleftarrow{\$} \{0, 1\}$
  $L \leftarrow \emptyset$
  $(\mathsf{sc}^*, \mathsf{state}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_0(\mathsf{usk}_0,\cdot),\mathcal{O}_1(\mathsf{usk}_1,\cdot)}(\mathsf{pp}_\Theta)$
    where oracle $\mathcal{O}_i$, for $i = 0, 1$, on input $\mathsf{sc}$:
      $L \leftarrow L \cup \{\mathsf{sc}\}$
      return $\mathsf{nym} \xleftarrow{\$} \mathsf{Gen}_\Theta(\mathsf{usk}_i, \mathsf{sc})$
  $\mathsf{nym}^* \xleftarrow{\$} \mathsf{Gen}_\Theta(\mathsf{usk}_b, \mathsf{sc}^*)$
  $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_0(\mathsf{usk}_0,\cdot),\mathcal{O}_1(\mathsf{usk}_1,\cdot)}(\mathsf{state}, \mathsf{nym}^*)$
  return 0, if $\mathsf{sc}^* \in L$
  return 1, if $b = b'$
  return 0
```

Fig. 1: Θ Unlinkability

3 Our Framework

We now introduce the formal framework for the Ω schemes with the mentioned additional capabilities. Our model is based upon prior work [6,13], but was heavily adjusted for our use-case. The main differences are made explicit in the definitions to ease readability.

To recap, a Ω scheme has the following participants: A *group-manager*, which decides which users participate in the group, i.e., can sign. Each *user*, can, after joining, generate signature on behalf of the group, while the *opener* can, in case of a dispute, pinpoint (or deny) the accountable user, and (un)link signatures.

To add an additional layer of privacy, we introduce the new notion opening-privacy definition. This definition says that an adversary does not learn which message belongs to a given signature, even if it can generate the opening key. This is achieved by letting the signing algorithm Sign_Ω return some additional verification information τ, which is not needed to open a signature, but only to verify it. Thus, if the opener does not receive the corresponding verification information τ, it cannot know which message a given signature σ protects. Additional changes to standard definitions are discussed within each definition.

3.1 Syntactic Framework

We now present the formal interfaces for our enhanced Ω.

Definition 4 (Group Signatures). *A group signature scheme Ω consists of PPT algorithms* $\{\mathsf{PGen}_{\Omega}, \mathsf{GKG}, \mathsf{OKG}, \mathsf{OKVf}, \mathsf{UKG}, \langle\mathsf{Join}; \mathsf{Iss}\rangle, \mathsf{Sign}_{\Omega}, \mathsf{Vf}_{\Omega}, \mathsf{Opn}, \mathsf{Jdg}, \mathsf{Lnk}, \mathsf{LnkJdg}, \mathsf{SLnk}, \mathsf{SLnkJdg}\}$ *such that:*

PGen_{Ω}. *This algorithm (executed by a trusted third party) generates the public parameters for the scheme:*

$$\mathsf{pp}_{\Omega} \overset{\$}{\leftarrow} \mathsf{PGen}_{\Omega}(\mathsf{pp}_{\mathsf{SYS}})$$

We assume that pp_{Ω} *is implicitly input to all other algorithms, while* $\mathsf{pp}_{\mathsf{SYS}}$ *are some global parameters.*

GKG. *This algorithm (executed by the group manager) generates the key pair of the group manager:*

$$(\mathsf{isk}, \mathsf{ipk}) \overset{\$}{\leftarrow} \mathsf{GKG}(\mathsf{pp}_{\Omega})$$

OKG. *This algorithm (executed by the opening manager) generates the key pair of the opener:*

$$(\mathsf{osk}, \mathsf{opk}) \overset{\$}{\leftarrow} \mathsf{OKG}(\mathsf{pp}_{\Omega})$$

OKVf. *This algorithm verifies whether a given opener public key* opk *corresponds to* osk*, where* $d \in \{0,1\}$*:*

$$d \overset{\$}{\leftarrow} \mathsf{OKVf}(\mathsf{osk}, \mathsf{opk})$$

UKG. *This algorithm (executed by a user) generates a key pair of a user:*

$$(\mathsf{usk}, \mathsf{upk}) \overset{\$}{\leftarrow} \mathsf{UKG}(\mathsf{pp}_{\Omega})$$

$\langle\mathsf{Join}; \mathsf{Iss}\rangle$. *The algorithms* Join *and* Iss *allow a user to join a group. As these are the only algorithms which require interaction, we denote this as a two-step protocol* $\langle\mathsf{Join}(\mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{ipk}); \mathsf{Iss}(\mathsf{isk}, \mathsf{ipk}, \mathsf{opk})\rangle$*, run between a user and the group manager, receives as input the secret user key* usk *(and the corresponding public key* upk*) from the user, the opening manager public key* opk*, as well as the issuer secret key* isk *(and the public key* ipk*). The only output is the secret user signing key* ssk *to the user, and* upk *to the issuer:*

$$\langle\mathsf{ssk}; \mathsf{upk}\rangle \overset{\$}{\leftarrow} \langle\mathsf{Join}(\mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{ipk}); \mathsf{Iss}(\mathsf{isk}, \mathsf{ipk}, \mathsf{opk})\rangle$$

Sign_{Ω}. *This algorithm (executed by the user) generates a signature σ, along with some verification information τ, on a message m w.r.t.* ipk*,* opk*,* ssk *and* usk *(and the corresponding public key* upk*):*

$$(\sigma, \tau) \overset{\$}{\leftarrow} \mathsf{Sign}_{\Omega}(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{ipk}, \mathsf{opk}, m)$$

Here, σ is needed for opening the signature, whereas τ is only needed to also verify the signature.

Vf_{Ω}. *This algorithm (executed by a verifier) verifies a signature σ on m w.r.t.* ipk*, τ, and* opk*, where* $d \in \{0,1\}$*:*

$$d \overset{\$}{\leftarrow} \mathsf{Vf}_{\Omega}(\mathsf{opk}, \mathsf{ipk}, \sigma, \tau, m)$$

Opn. *This algorithm (executed by the opening manager) generates a proof* π_{opener} *(to be used by* Jdg*) which either reveals the accountable party or shows that the owner of* upk' *is (not) the creator of a signature σ w.r.t.* ipk *and* opk*:*

$$(\pi_{\mathsf{opener}}, \mathsf{upk}) \overset{\$}{\leftarrow} \mathsf{Opn}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, \sigma, \mathsf{upk}')$$

Note, the opener does neither receive m nor τ. If $\mathsf{upk}' = \perp$*, this algorithm behaves as a standard opening, i.e., it finds* upk *(if possible).*

Jdg. *This algorithm (executed by whatever party) decides whether* π_{opener} *is a valid proof that owner of* upk *is really accountable ($b = 1$), or not ($b = 0$), for the signature σ on message m, w.r.t.* ipk*, τ and* opk*, where* $d \in \{0,1\}$*:*

$$d \leftarrow \mathsf{Jdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{opener}}, \mathsf{upk}, \sigma, \tau, m, b)$$

SOpn. *This algorithm (executed by a user) generates a proof* π_{signer} *(to be used by* SJdg*), proving whether the holder of* usk *is accountable for a signature* σ *on message* m*, or not, on input of* opk*,* τ*,* ipk*,* ssk*, and* usk *(and the corresponding public key* upk*):*

$$\pi_{\mathsf{signer}} \xleftarrow{\$} \mathsf{SOpn}(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{ipk}, \sigma, \tau, m)$$

SJdg. *This deterministic algorithm (executed by whatever party) decides whether the proof* π_{signer} *shows that the owner* upk *is accountable (*$b = 1$*), or not (*$b = 0$*), for the values (*σ, τ*) on a message* m *w.r.t.* ipk *and* opk*, where* $d \in \{0, 1\}$*:*

$$d \leftarrow \mathsf{SJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{signer}}, \mathsf{upk}, \sigma, \tau, m, b)$$

Lnk. *This algorithm (executed by the opening manager) allows to generate a proof whether two*[5] *signatures* σ_0 *and* σ_1 *stem from the same signer or not w.r.t. to* osk *(and the corresponding public key* opk*) and* ipk*:*

$$\pi_{\mathsf{link}} \xleftarrow{\$} \mathsf{Lnk}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, \sigma_0, \sigma_1)$$

LnkJdg. *This deterministic algorithm (executed by whatever party) decides whether* π_{link} *is a valid proof that two signatures* σ_0 *and* σ_1 *stem from the same signer (*$b = 1$*) or not (*$b = 0$*), where* $d \in \{0, 1\}$*:*

$$d \leftarrow \mathsf{LnkJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{link}}, \sigma_0, \sigma_1, \tau_0, \tau_1, m_0, m_1, b)$$

SLnk. *This algorithm (executed by a user) allows to generate a proof whether two signatures* σ_0 *and* σ_1 *(along with* m_0*,* m_1*,* τ_0 *and* τ_1*) stem from it or not w.r.t. to* usk *(and the corresponding public key* upk*),* opk *and* ipk*:*

$$\pi_{\mathsf{linku}} \xleftarrow{\$} \mathsf{SLnk}(\mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{ipk}, \sigma_0, \sigma_1, \tau_0, \tau_1, m_0, m_1)$$

$\mathsf{SLnkJdg}$. *This deterministic algorithm (executed by whatever party) decides whether* π_{linku} *is a valid proof that two signatures* σ_0 *and* σ_1 *(along with* τ_0 *and* τ_1*, as well as the message* m_0 *and* m_1*) stem from the same signer (*$b = 1$*) or not (*$b = 0$*), where* $d \in \{0, 1\}$*:*

$$d \leftarrow \mathsf{SLnkJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{linku}}, \sigma_0, \sigma_1, \tau_0, \tau_1, m_0, m_1, b)$$

Correctness. Informally, correctness requires that, when called on an honestly generated inputs, no algorithm halts in an error state. Furthermore, honestly generated signatures can always be verified and opened, any honestly computed opening proof verifies correctly, and given honestly generated and consistent inputs, all signature linking verifications verify correctly. A formalization is straightforward, and thus omitted here.

3.2 Security Framework

Subsequently, we present the security framework.

Most of these definitions are based on existing work [6,13], but are altered to account for our use-case. For example, we need to limit access to the linking oracles to avoid trivial attacks on anonymity. To avoid confusion, we stress that due to the additional linking capabilities of our Ω, the adversary must be able to return two signatures or messages in some of the extended definitions. This, however, still captures the "standard" definitions, as the adversary can always return the same values twice.

Security Framework. Subsequently, we present the formal security framework our constructions are proven secure in. Most of these definitions are standard, but are altered to account for our use-case. Note, honest (and later to-be-corrupted) users can be simulated by the adversary itself.

To recap, we do not allow for corruptions, as otherwise, due to self-opening, anonymity breaks down. Likewise, we need to limit access to the linking oracles to avoid trivial attacks on anonymity. Moreover, we introduce an additional property required in our setting. Namely, the new opening-privacy definition says that an adversary does not learn which message belongs to a given signature, even if it can generate the opening key osk, if it does not receive the corresponding verification information τ.

[5] Here and in the following, we restrict ourselves to *two* signatures. However, extending the interfaces of Lnk and SLnk, constructions, definitions, and proofs to an arbitrary number of signatures is straightforward, but introduces notational complexity without providing further insights.

```
**Experiment** $\mathsf{Non\text{-}Frameability}^{\Omega}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp}_\Omega \xleftarrow{\$} \mathsf{PGen}_\Omega(1^\lambda)$
  $(\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{UKG}(\mathsf{pp}_\Omega)$
  $\mathcal{Q} = \mathcal{R} \leftarrow \emptyset$
  $(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*, \sigma_0^*, \sigma_1^*, \tau_0^*, \tau_1^*, m_0^*, m_1^*) \xleftarrow{\$} \mathcal{A}^{\langle \mathsf{Join}(\cdot,\cdot);\mathcal{A}\rangle', \mathsf{Sign}'_\Omega(\cdot,\cdot,\cdot)}_{\mathsf{SOpn}(\cdot,\cdot,\cdot,\cdot,\cdot), \mathsf{SLnk}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{upk})$
    where oracle $\langle \mathsf{Join}; \mathcal{A}\rangle'$ on input $\mathsf{opk}'$, $\mathsf{ipk}'$:
      return $\perp$, if $(\mathsf{opk}', \mathsf{ipk}', \cdot) \in \mathcal{Q}$
      let $\langle \mathsf{ssk}; \cdot\rangle \xleftarrow{\$} \langle \mathsf{Join}(\mathsf{usk}, \mathsf{upk}, \mathsf{opk}', \mathsf{ipk}'); \mathcal{A}\rangle$
      if $\mathsf{ssk} \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk})\}$
    where oracle $\mathsf{Sign}'_\Omega$ on input $\mathsf{opk}'$, $\mathsf{ipk}'$, $m$:
      if $(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}) \notin \mathcal{Q}$, return $\perp$
      let $(\sigma, \tau) \xleftarrow{\$} \mathsf{Sign}_\Omega(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{ipk}', \mathsf{opk}', m)$
      let $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathsf{opk}', \mathsf{ipk}', \sigma, \tau, m)\}$
      return $(\sigma, \tau)$
  return 1, if
    $((\mathsf{opk}^*, \mathsf{ipk}^*, \sigma_0^*, \tau_0^*, m_0^*) \notin \mathcal{R} \wedge$
      $(\mathsf{Jdg}(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*, \mathsf{upk}, \sigma_0^*, \tau_0^*, m_0^*, 1) = 1 \vee$
      $\mathsf{SJdg}(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*, \mathsf{upk}, \sigma_0^*, \tau_0^*, m_0^*, 1) = 1)) \vee$
    $((\mathsf{opk}^*, \mathsf{ipk}^*, \sigma_0^*, \tau_0^*, m_0^*) \in \mathcal{R} \ \wedge \ (\mathsf{opk}^*, \mathsf{ipk}^*, \sigma_1^*, \tau_1^*, m_1^*) \notin \mathcal{R} \wedge$
    $(\mathsf{LnkJdg}(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*, \sigma_0^*, \sigma_1^*, \tau_0^*, \tau_1^*, m_0^*, m_1^*, 1) = 1 \vee$
    $\mathsf{SLnkJdg}(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*, \sigma_0^*, \sigma_1^*, \tau_0^*, \tau_1^*, m_0^*, m_1^*, 1) = 1))$
```

Fig. 2: Ω Non-Frameability

Non-Frameability. The property of non-frameability says that even a corrupt issuer working together with a corrupt opening manager cannot blame a honest user for a signature it did not create for any group, even if created by the adversary. In our setting, this must even hold for linking signatures, i.e., if a honest user did not create both signatures in question, the adversary cannot create a proof which links those signatures.

In more detail, the adversary can generate all key-pairs but a user's one. Thus, this key pair is honestly chosen. The user's public key is given to the adversary. Then, the adversary gains access to a join oracle (where it again chooses all public keys w.r.t. to the groups), all signing related oracles, and access to all self-open and self-linking oracles with arbitrary input. The adversary then wins, if it can output an issuer public-key (ipk^*), opener public-key (opk^*), a bogus proof π^*, along with two messages (m_0^* and m_1^*), two signatures (σ_0^* and σ_1^*), and two auxiliary verification values (τ_0 and τ_1) which either point to the honest user's public key (even though that signature has never been created w.r.t. to returned values) or the forged signature becomes linkable to a signature actually created by the signer.

Definition 5 (Ω Non-Frameability). *An Ω is non-frameable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν, such that:*

$$\Pr[\mathsf{Non\text{-}Frameability}^{\Omega}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is defined in Figure 2.

Anonymity. Anonymity guarantees that no party, but the opening manager, can decide which party has issued a given signature. This must even hold for corrupt issuers, while the adversary is not allowed to open signatures from the challenge oracle due to obvious reasons, i.e., trivial attacks. This is formalized by challenging the adversary with a left-or-right oracle which either signs in the name of a first or a second user. We stress that if all users, but one, work together, anonymity cannot hold. This, however, is true for all Ω schemes.

In more detail, the challenger generates two user key-pairs. The challenger draws a random bit $b \xleftarrow{\$} \{0, 1\}$, yet also generates the key pair (osk and opk) for an opener honestly. The adversary receives the users' public keys and opk. Its goal is to guess b. The adversary gains access to a join oracle (where it can chose ipk and opk), signing oracle, a left-or-right signing oracle, an opening oracle, a self-opening oracle, a linking oracle, and a self-linking oracle. However, the signing oracle also keeps track which signatures have been created (which is later used in the linking oracles to avoid trivial "transitivity attacks", i.e., to avoid that the adversary uses signatures from several sources to form a linked chain)

```
**Experiment** $\mathsf{Anonymity}^{\Omega}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp}_{\Omega} \xleftarrow{\$} \mathsf{PGen}_{\Omega}(1^{\lambda})$
  $b \xleftarrow{\$} \{0,1\}$
  $(\mathsf{osk}, \mathsf{opk}) \xleftarrow{\$} \mathsf{OKG}(\mathsf{pp}_{\Omega})$
  $\forall i \in \{0,1\}$, let $(\mathsf{usk}^i, \mathsf{upk}^i) \xleftarrow{\$} \mathsf{UKG}(\mathsf{pp}_{\Omega})$
  $\mathcal{Q}_0 = \mathcal{Q}_1 = \mathcal{R} = \mathcal{T} \leftarrow \perp$
  $a \xleftarrow{\$} \mathcal{A}^{\langle \mathsf{Join}(\cdot,\cdot,\cdot);\mathcal{A}\rangle', \mathsf{Sign}'_{\Omega}(\cdot,\cdot,\cdot), \mathsf{LoRSig}(\cdot,\cdot,b), \mathsf{Opn}'(\cdot,\cdot,\cdot)}_{\mathsf{SOpn}'(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot), \mathsf{Lnk}'(\cdot,\cdot,\cdot), \mathsf{SLnk}'(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{opk}, \mathsf{upk}^0, \mathsf{upk}^1)$
    where oracle $\langle \mathsf{Join}; \mathcal{A}\rangle'$ on input $i$, $\mathsf{opk}'$, $\mathsf{ipk}'$:
      return $\perp$, if $i \notin \{0,1\} \ \vee \ (\mathsf{opk}', \mathsf{ipk}', \cdot, \mathsf{usk}^i, \mathsf{upk}^i) \in \mathcal{Q}_i$
      let $\langle \mathsf{ssk}; \cdot \rangle \xleftarrow{\$} \langle \mathsf{Join}(\mathsf{usk}^i, \mathsf{upk}^i, \mathsf{opk}', \mathsf{ipk}'); \mathcal{A} \rangle$
      if $\mathsf{ssk} \neq \perp$, let $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}, \mathsf{usk}^i, \mathsf{upk}^i)\}$
    where oracle $\mathsf{Sign}'_{\Omega}$ on input $i$, $\mathsf{ipk}'$, $\mathsf{opk}'$, $m$:
      if $i \notin \{0,1\} \ \vee \ (\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}^i, \mathsf{usk}^i, \mathsf{upk}^i) \notin \mathcal{Q}_i$, return $\perp$
      let $(\sigma, \tau) \xleftarrow{\$} \mathsf{Sign}_{\Omega}(\mathsf{ssk}^i, \mathsf{usk}^i, \mathsf{upk}^i, \mathsf{opk}', \mathsf{ipk}', m)$
      let $\mathcal{T} \leftarrow \mathcal{T} \cup \{(\mathsf{opk}', \mathsf{ipk}', \sigma, \tau, m, i)\}$
      return $(\sigma, \tau)$
    where oracle $\mathsf{LoRSig}$, on input $\mathsf{ipk}'$, $m$, $b$:
      if $(\mathsf{opk}, \mathsf{ipk}', \mathsf{ssk}^j, \mathsf{usk}^j, \mathsf{upk}^j) \notin \mathcal{Q}_j$ for a $j \in \{0,1\}$, return $\perp$
      let $(\sigma, \tau) \xleftarrow{\$} \mathsf{Sign}_{\Omega}(\mathsf{ssk}^b, \mathsf{usk}^b, \mathsf{upk}^b, \mathsf{opk}, \mathsf{ipk}', m)$
      let $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathsf{opk}, \mathsf{ipk}', \sigma, \tau, m)\}$
      return $(\sigma, \tau)$
    where oracle $\mathsf{Opn}'$, on input $\mathsf{ipk}'$, $\sigma$, $\mathsf{upk}'$:
      if $(\mathsf{opk}, \mathsf{ipk}', \sigma, \cdot, \cdot) \in \mathcal{R} \ \wedge \ \mathsf{upk}' \in \{\mathsf{upk}^0, \mathsf{upk}^1, \perp\}$, return $\perp$
      return $\mathsf{Opn}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}', \sigma, \mathsf{upk}')$
    where oracle $\mathsf{SOpn}'$, on input $i$, $\mathsf{opk}'$, $\mathsf{ipk}'$, $\sigma$, $\tau$, $m$
      if $(\mathsf{opk}', \mathsf{ipk}', \sigma, \tau, m) \in \mathcal{R} \ \vee \ i \notin \{0,1\} \ \vee$
        $(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}^i, \mathsf{usk}^i, \mathsf{upk}^i) \notin \mathcal{Q}_i$, return $\perp$,
      return $\mathsf{SOpn}(\mathsf{ssk}^i, \mathsf{usk}^i, \mathsf{upk}^i, \mathsf{opk}', \mathsf{ipk}', \sigma, m)$
    where oracle $\mathsf{Lnk}'$, on input $\mathsf{ipk}'$, $\sigma_0$, $\sigma_1$
      if $(\mathsf{opk}, \mathsf{ipk}', \sigma_j, \cdot, \cdot) \in \mathcal{R} \ \wedge$
        ”$'(\mathsf{opk}, \mathsf{ipk}', \sigma_{1-j}, \cdot, \cdot, \cdot) \in \mathcal{T}$ for a $j \in \{0,1\}$, return $\perp$
      return $\mathsf{Lnk}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}', \sigma_0, \sigma_1)$
    where oracle $\mathsf{SLnk}'$, on input $i$, $\mathsf{opk}'$, $\mathsf{ipk}'$, $\sigma_0$, $\sigma_1$, $\tau_0$, $\tau_1$, $m_0$, $m_1$, $b''$
      if $(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}^i, \mathsf{usk}^i, \mathsf{upk}^i) \notin \mathcal{Q}_i \ \vee \ i \notin \{0,1\}$, return $\perp$
      if $(\mathsf{opk}', \mathsf{ipk}', \sigma_j, \tau_j, m_j) \in \mathcal{R} \ \wedge$, return $\perp$
        $(\mathsf{opk}', \mathsf{ipk}', \sigma_{1-j}, \tau_{1-j}, m_{1-j}) \in \mathcal{T}$ for a $j \in \{0,1\}$
      let $i \leftarrow b' \oplus b''$, if $(\mathsf{opk}', \mathsf{ipk}', \sigma_0, \tau_0, m_0, b') \in \mathcal{R} \ \wedge$
        $(\mathsf{opk}', \mathsf{ipk}', \sigma_1, \tau_1, m_1, b') \in \mathcal{R}$
      return $\mathsf{SLnk}(\mathsf{usk}^i, \mathsf{upk}^i, \mathsf{opk}', \mathsf{ipk}', \sigma_0, \sigma_1, \tau_0, \tau_1, m_0, m_1)$
  return 1, if $a = b$
  return 0
```

Fig. 3: Ω Anonymity

in a list $\mathcal{T}$. For the same reason, the left-or-right oracle keeps generated signatures (it only generates signatures for user b and the challenge opk) in a list $\mathcal{R}$. The opening-oracle, also to avoid trivial attacks, does not open signatures generated from the left-or-right oracle (known due to the list $\mathcal{T}$). The same is true for the self-opening and self-linking oracles, which, however, allow the adversary to input arbitrary opks. Moreover, as already explained, the linking oracles prohibit linking signatures generated from different signing oracles to avoid trivially leaking the bit b to the adversary. However, in the self-linking oracle, we allow the adversary to receive a proof for signatures all coming from the left-or-right oracle, as this may leak information as well.

```
**Experiment** $\mathsf{Traceability}^{\Omega}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp}_{\Omega} \xleftarrow{\$} \mathsf{PGen}_{\Omega}(1^{\lambda})$
  $(\mathsf{isk}, \mathsf{ipk}) \xleftarrow{\$} \mathsf{GKG}(\mathsf{pp}_{\Omega})$
  $(\mathsf{osk}, \mathsf{opk}) \xleftarrow{\$} \mathsf{OKG}(\mathsf{pp}_{\Omega})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m_0^*, \sigma_0^*, \tau_0^*, m_1^*, \sigma_1^*, \tau_1^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Opn}(\cdot,\cdot,\cdot,\cdot,\cdot), \langle \mathcal{A}; \mathsf{Iss}(\cdot) \rangle', \mathsf{Lnk}(\cdot,\cdot,\cdot)}(\mathsf{opk}, \mathsf{ipk})$
    where oracle $\langle \mathcal{A}; \mathsf{Iss}(\cdot) \rangle'$ on input $\mathsf{opk}'$:
      let $(\cdot; \mathsf{upk}) \xleftarrow{\$} \langle \mathcal{A}; \mathsf{Iss}(\mathsf{isk}, \mathsf{ipk}, \mathsf{opk}') \rangle$
      if $\mathsf{upk} \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{opk}', \mathsf{upk})\}$
  return 0, if $\mathsf{Vf}_{\Omega}(\mathsf{opk}, \mathsf{ipk}, \sigma_j^*, \tau_j^*, m_j^*) = 0$ for a $j \in \{0, 1\}$
  let $(\pi_{\mathsf{opener}}^i, \mathsf{upk}^i) \xleftarrow{\$} \mathsf{Opn}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, \sigma_i^*, \perp)$ for $i \in \{0, 1\}$
  if $\mathsf{upk}^0 = \mathsf{upk}^1$:
    let $\pi_{\mathsf{link}} \xleftarrow{\$} \mathsf{Lnk}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, \sigma_0, \sigma_1)$
  else:
    $\pi_{\mathsf{link}} \leftarrow \perp$ otherwise.
  return 1, if $\mathsf{Jdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{opener}}^0, \mathsf{upk}^0, \sigma_0^*, \tau_0^*, m_0^*, 1) \neq 1 \vee$
    $\mathsf{Jdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{opener}}^1, \mathsf{upk}^1, \sigma_1^*, \tau_1^*, m_1^*, 1) \neq 1 \vee$
    $(\mathsf{Jdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{opener}}^j, \mathsf{upk}^j, \sigma_j^*, \tau_j^*, m_j^*, 1) = 1$
        $\wedge\ (\mathsf{opk}, \mathsf{upk}^j) \notin \mathcal{Q}$ for a $j \in \{0, 1\}) \vee$
    $(\pi_{\mathsf{link}} \neq \perp \ \wedge \ \mathsf{LnkJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_{\mathsf{link}}, \sigma_0^*, \sigma_1^*, \tau_0^*, \tau_1^*, m_0^*, m_1^*, 1) \neq 1)$
  return 0
```

Fig. 4: Ω Traceability

Definition 6 (Ω Anonymity). *An Ω is anonymous, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν, such that:*

$$\left| \Pr[\mathsf{Anonymity}^{\Omega}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right| \leq \nu(\lambda)$$

The corresponding experiment is defined in Figure 3.

Traceability. Traceability requires that no adversary can generate a valid signature which cannot be traced to a specific joined user. In our case, this is also true for linking, i.e., if two signatures stem from the same signer, an honest opener can link them, and the adversary cannot claim otherwise.

In more detail, the challenger generates the issuer and opening keys honestly. The adversary's goal is to output two messages (m_0^* and m_1^*), two signatures (σ_0^* and σ_1^*), and two auxiliary verification values (τ_0 and τ_1) which either cannot be opened or linked.

Definition 7 (Ω Traceability). *An Ω is traceable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν, such that:*

$$\Pr[\mathsf{Traceability}^{\Omega}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is defined in Figure 4.

Trace-Soundness. Trace-Soundness requires that a signature can only be opened in an unambiguous way, i.e., an adversary cannot claim authorship of a signature it did not create. This must even be true, if the adversary can create all keys in the system. Thus, no oracles are needed. In our case, this also means that the adversary cannot create two conflicting proofs for linking signatures. This extends the definition by Sakai et al. [34].

Definition 8 (Ω Trace-Soundness). *An Ω is trace-sound, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν, such that:*

$$\Pr[\mathsf{Trace\text{-}Soundness}^{\Omega}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is defined in Figure 5.

```
$\textbf{Experiment}\ \mathsf{Trace\text{-}Soundness}^{\Omega}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp}_{\Omega} \xleftarrow{\$} \mathsf{PGen}_{\Omega}(1^{\lambda})$
  $(\mathsf{isk}^*, \mathsf{osk}^*, \pi^*, \mathsf{upk}^*_0, \mathsf{upk}^*_1, \pi^*_0, \pi^*_1, \sigma^*_0, \sigma^*_1, \tau^*_0, \tau^*_1, m^*_0, m^*_1) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}_{\Omega})$
  return 1, if $\exists \mathsf{Alg}_1, \mathsf{Alg}_2 \in \{\mathsf{Jdg}, \mathsf{SJdg}\}, \exists \mathsf{Alg}_3, \mathsf{Alg}_4 \in \{\mathsf{LnkJdg}, \mathsf{SLnkJdg}\}$:
    $\big(\mathsf{upk}^*_0 \neq \mathsf{upk}^*_1 \wedge$
    $(\mathsf{SJdg}(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*_0, \mathsf{upk}^*_0, \sigma^*_0, \tau^*_0, m^*_0, 0) = 1 \wedge$
      $\mathsf{SJdg}(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*_1, \mathsf{upk}^*_0, \sigma^*_0, \tau^*_0, m^*_0, 1) = 1)\ \vee$
    $(\mathsf{Alg}_1(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*_0, \mathsf{upk}^*_0, \sigma^*_0, \tau^*_0, m^*_0, 1) = 1 \wedge$
      $\mathsf{Alg}_2(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*_1, \mathsf{upk}^*_1, \sigma^*_0, \tau^*_0, m^*_0, 1) = 1) \vee$
    $(\mathsf{Alg}_3(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*_0, \sigma^*_0, \sigma^*_1, \tau^*_0, \tau^*_1, m^*_0, m^*_1, 1) = 1 \wedge$
      $\mathsf{Alg}_4(\mathsf{opk}^*, \mathsf{ipk}^*, \pi^*_1, \sigma^*_0, \sigma^*_1, \tau^*_0, \tau^*_1, m^*_0, m^*_1, 0) = 1)\big)$
  return 0
```

Fig. 5: Ω Trace-Soundness

```
$\textbf{Experiment}\ \mathsf{Opening\text{-}Privacy}^{\Omega}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp}_{\Omega} \xleftarrow{\$} \mathsf{PGen}_{\Omega}(1^{\lambda})$
  $b \xleftarrow{\$} \{0,1\}$
  $(\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{UKG}(\mathsf{pp}_{\Omega})$
  $\mathcal{Q} \leftarrow \emptyset$
  $a \xleftarrow{\$} \mathcal{A}^{\langle \mathsf{Join}(\cdot,\cdot);\mathcal{A}\rangle', \mathsf{Sign}'_{\Omega}(\cdot,\cdot,\cdot), \mathsf{LoRSig}(\cdot,\cdot,\cdot,\cdot), \mathsf{SOpn}(\cdot,\cdot,\cdot,\cdot,\cdot), \mathsf{SLnk}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{upk})$
    where oracle $\langle \mathsf{Join}; \mathcal{A}\rangle'$ on input $\mathsf{opk}'$, $\mathsf{ipk}'$:
      if $(\mathsf{opk}', \mathsf{ipk}', \cdot) \in \mathcal{Q}$, return $\perp$
      let $\langle \mathsf{ssk}; \cdot \rangle \xleftarrow{\$} \langle \mathsf{Join}(\mathsf{usk}, \mathsf{upk}, \mathsf{opk}', \mathsf{ipk}'); \mathcal{A}\rangle$
      if $\mathsf{ssk} \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk})\}$
    where oracle $\mathsf{Sign}'_{\Omega}$ on input $\mathsf{opk}'$, $\mathsf{ipk}'$, $m$:
      if $(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}) \notin \mathcal{Q}$, return $\perp$
      return $\mathsf{Sign}_{\Omega}(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{ipk}', \mathsf{opk}', m)$
    where oracle $\mathsf{LoRSig}$ on input $\mathsf{opk}'$, $\mathsf{ipk}'$, $m_0$, $m_1$:
      if $(\mathsf{opk}', \mathsf{ipk}', \mathsf{ssk}) \notin \mathcal{Q}$, return $\perp$
      let $(\sigma, \tau) \xleftarrow{\$} \mathsf{Sign}_{\Omega}(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{ipk}', \mathsf{opk}', m_b)$
      return $\sigma$
  return 1, if $a = b$
```

Fig. 6: Ω Opening-Privacy

Opening-Privacy. Opening-Privacy requires that a signature σ does not leak the message m to which is belongs. We define it in such a way that the adversary, even if it can generate opk, cannot decide which message m a given signature σ protects, if it does not receive the corresponding verification information τ. This is formalized by a left-or-right signing oracle, which either signs m_0 or m_1 and does not return the corresponding τ, but only σ.

In more detail, the challenger draws a bit $b \xleftarrow{\$} \{0,1\}$ and generates a single user key-pair. The user's public key is handed to the adversary. All other keys are generated by the adversary. Moreover, the adversary gains access to a joining oracle, a signing oracle, a self-open oracle and a self-link oracle, as well as an left-or-right oracle. All oracles behave as normal, but the left-or-right oracle only returns a signature for message m_b, while the adversary can input two messages (m_0 and m_1) of its choice. Thus, the oracle either signs m_0 or m_1, but does not return the corresponding verification information τ, but only σ. The adversary wins, if it can guess b correctly.

Definition 9 (Ω Opening-Privacy). *An Ω is opening-private, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν, such that:*

$$\left| \Pr[\mathsf{Opening\text{-}Privacy}^{\Omega}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right| \leq \nu(\lambda)$$

The corresponding experiment is defined in Figure 6.

We conclude this section with a final definition:

Definition 10 (Secure Ω). *We call a Ω secure, if it is correct, anonymous, non-frameable, traceable, trace-sound, and opening-private.*

4 Our Generic Construction

We next present a black-box construction fulfilling the definitions presented above. However, to ease understanding, we give a high-level idea beforehand.

Intuition. Our approach is based on the efficient encrypt-and-proof paradigm by Bellare et al. [6]. In a nutshell, the secret signing key of the user is a signature on the key opk of the opening manager and a Θ public key. For signing, the user draws a random string pad ("padding"), generates the corresponding pseudonym from the concatenated message/padding, and encrypts its public key towards the opener. The auxiliary opening information τ is exactly the padding. It then generates a proof that everything was calculated correctly, with the message and padding as a label. Verification is thus checking the validity of the generated proof. Opening works as follows: The opener decrypts the ciphertext and proves that it did so correctly and knows osk. The same idea is also true for linking: the opener proves that both ciphertexts contain the same upk (without revealing it). In the denying case, the idea is the same, but the opener proves inequality. From the user's side, however, things change a bit. A user either proves that it knows the secret pseudonym key usk for self-opening and self-linking, or that the secret keys are different. Thus, all judges simply verify proofs, boiling down to a handful of exponentiations.

Additional Conventions. For the zero-knowledge proofs, we use the notation introduced by Camenisch and Stadler [17], i.e., instead of Prv_{Ξ} we write $\mathsf{ZKP}[(a,b) : x = g^a h^b \wedge y = g^b]$ to denote a Ξ of a, b such that the relation on the right hand side is satisfied; All values not specified in the first parentheses are public.

In case that one or more of the building blocks are using common reference strings (CRS), or random oracles (RO), the resulting gets access to all these CRS or ROs as well, where we assume that the oracles for the different building blocks are fully independent from each other.

Parameter Generation. The public parameters pp_{Ω} consist of the public parameters of all the building blocks. That is, $\mathsf{PGen}_{\Omega}(1^{\lambda})$ behaves as follows: After generating potential system parameters $\mathsf{pp}_{\mathsf{SYS}} \xleftarrow{\$} \mathsf{PGen}_{\mathsf{SYS}}(1^{\lambda})$ for some global parameter generation algorithm, it computes $\mathsf{pp}_{\Sigma} \xleftarrow{\$} \mathsf{PGen}_{\Sigma}(\mathsf{pp}_{\mathsf{SYS}})$, $\mathsf{pp}_{\Pi} \xleftarrow{\$} \mathsf{PGen}_{\Pi}(\mathsf{pp}_{\mathsf{SYS}})$, and $\mathsf{pp}_{\Theta} \xleftarrow{\$} \mathsf{PGen}_{\Theta}(\mathsf{pp}_{\mathsf{SYS}})$. Finally, the algorithm outputs $\mathsf{pp}_{\Omega} \leftarrow (\mathsf{pp}_{\mathsf{SYS}}, \mathsf{pp}_{\Sigma}, \mathsf{pp}_{\Pi}, \mathsf{pp}_{\Theta})$.

Key Generation. The different parties in the system generate their secret and public key material as follows:

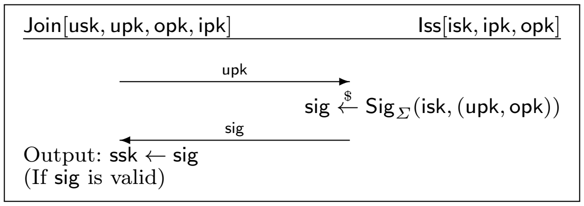
The group manager generates a signing and verification key pair of the signature scheme, i.e., $\mathsf{GKG}(\mathsf{pp}_{\Omega})$ first extracts the parameters pp_{Σ} of the signature scheme from the overall public parameters. It then generates $(\mathsf{isk}, \mathsf{ipk}) \xleftarrow{\$} \mathsf{KG}_{\Sigma}(\mathsf{pp}_{\Sigma})$.

The opener generates a decryption and encryption key pair of an encryption scheme, i.e., $\mathsf{OKG}(\mathsf{pp}_{\Omega})$ first extracts the parameters pp_{Π} of the encryption scheme from the overall public parameters. It then generates $(\mathsf{osk}, \mathsf{opk}) \xleftarrow{\$} \mathsf{KG}_{\Pi}(\mathsf{pp}_{\Pi})$. The key verification algorithm OKVf internally simply executes the corresponding algorithm of the encryption scheme, i.e., KVf_{Π}.

The key pair of a user is given as the secret key of a pseudonym system and a pseudonym for a fixed scope. That is, $\mathsf{UKG}(\mathsf{pp}_{\Omega})$ first extracts the parameters pp_{Θ} from the overall public parameters, and computes $\mathsf{usk} \xleftarrow{\$} \mathsf{KG}_{\Theta}(\mathsf{pp}_{\Theta})$. It then computes a pseudonym for scope $\mathtt{setup}$ as $\mathsf{upk} \leftarrow \mathsf{Gen}_{\Theta}(\mathsf{usk}, \mathtt{setup})$, where we assume that $\mathtt{setup}$ is a special string that is solely used for the purpose of generating and later proving this pseudonym.

Issuance. When a user joins a group, the user simply receives a signature on its public key and the group opener key. The flow of this simple protocol (using the notation from the algorithms above) is depicted in Protocol 1.

Signing. In order to sign a message m, a user encrypts its public key, computes a pseudonym to a padded version of the message, and then computes a signature proof of knowledge showing that these computations were done correctly, and that the new pseudonym was derived from a secret key for which the user also possesses a valid signature from the issuer on the corresponding pseudonym for the scope $\mathtt{setup}$. More formally, the algorithm $\mathsf{Sign}_{\Omega}(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{ipk}, \mathsf{opk}, m)$ works as follows:



Prot. 1: The generic issuance protocol $\langle\mathsf{Join};\mathsf{Iss}\rangle$

1. Draw a 2λ-bit padding pad. Set $\mathsf{nym}_{m\|\mathsf{pad}} \leftarrow \mathsf{Gen}_\Theta(\mathsf{usk}, h)$, where $h = \mathcal{H}(m\|\mathsf{pad})$.
2. Set $(e; r) \xleftarrow{\$} \mathsf{Enc}(\mathsf{opk}, \mathsf{upk})$.
3. Compute the following signature proof of knowledge:

$$\begin{aligned}\pi_s \xleftarrow{\$} \mathsf{ZKP}\Big[&(\mathsf{usk}, \mathsf{upk}, \mathsf{ssk}, r) : \mathsf{upk} = \mathsf{Gen}_\Theta(\mathsf{usk}, \mathtt{setup}) \;\wedge \\ &\mathsf{nym}_{m\|\mathsf{pad}} = \mathsf{Gen}_\Theta(\mathsf{usk}, h) \;\wedge\; e = \mathsf{Enc}(\mathsf{opk}, \mathsf{upk}; r) \;\wedge \\ &\mathsf{Vf}_\Sigma(\mathsf{ipk}, \mathsf{ssk}, (\mathsf{upk}, \mathsf{opk})) = 1\Big](h, \mathsf{ctx})\,,\end{aligned}$$

   where $\mathsf{ctx} = (\mathsf{pp}_\Omega, h, e, \mathsf{opk}, \mathsf{ipk}, \mathsf{nym}_{m\|\mathsf{pad}})$ (signing ctx essentially rules out the malleability problems identified in [9]).
4. Output $(\sigma, \tau) \leftarrow ((e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad})$.

To verify a signature, $\mathsf{Vf}_\Omega(\mathsf{opk}, \mathsf{ipk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, m)$ first recomputes the scope ctx. It then checks whether $\mathsf{pad} \in \{0,1\}^{2\lambda}$, and whether $h = \mathcal{H}(m\|\mathsf{pad})$. Output 0, if this is not the case. It then verifies π_s with respect to ctx, and outputs the result of this verification.

Inspection. In order to open a signature σ, the opener checks the validity of π_s, and then simply decrypts e, finally returning the revealed public key of the user together with a zero-knowledge proof of knowledge showing the correctness of the decryption, or proves that the decryption is different from the target key upk'.

In more detail, $\mathsf{Opn}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, \sigma, \mathsf{upk}')$ performs the following steps:

1. Verify π_s (pad is not required to verify the proof).
2. Compute $\widehat{\mathsf{upk}} \leftarrow \mathsf{Dec}(\mathsf{osk}, e)$.
3. If $\mathsf{upk}' = \perp$:
   (a) Compute the following zero-knowledge proof of knowledge:

$$\pi_o \xleftarrow{\$} \mathsf{ZKP}\left[(\mathsf{osk}) : \mathsf{OKVf}(\mathsf{osk}, \mathsf{opk}) = 1 \wedge \widehat{\mathsf{upk}} = \mathsf{Dec}(\mathsf{osk}, e)\right](\mathsf{ctx})\,,$$

   where $\mathsf{ctx} = (\mathsf{pp}_\Omega, \mathsf{opk}, \mathsf{ipk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \widehat{\mathsf{upk}})$.
   (b) Output $(\pi_{\mathsf{opener}}, \mathsf{upk}) \leftarrow (\pi_o, \widehat{\mathsf{upk}})$.
4. Else, if $\mathsf{upk}' \neq \perp$:
   (a) Set $b = 1$, if $\widehat{\mathsf{upk}} = \mathsf{upk}'$ and $b = 0$ otherwise.
   (b) Compute the following zero-knowledge proof of knowledge:

$$\pi_o \xleftarrow{\$} \mathsf{ZKP}\left[(\mathsf{osk}) : \mathsf{OKVf}(\mathsf{osk}, \mathsf{opk}) = 1 \wedge \mathsf{upk}' \sim \mathsf{Dec}(\mathsf{osk}, e)\right](\mathsf{ctx})\,,$$

   where $\sim \in \{=, \neq\}$, depending on whether the user with identity upk' is $(b = 1)$ or is not $(b = 0)$ accountable for the given signature, and where

$$\mathsf{ctx} = (\mathsf{pp}_\Omega, \mathsf{opk}, \mathsf{ipk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{upk}', b).$$

   (c) Output $(\pi_o, \perp)$.

To verify whether a user is really accountable for a signature σ, algorithm $\mathsf{Jdg}(\mathsf{opk}, \mathsf{ipk}, \pi_o, \mathsf{nym}', (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, m, b)$ behaves as follows:

1. Output 0, if $\mathsf{Vf}_\Omega(\mathsf{ipk}, \mathsf{opk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, m) = 0$.
2. Output 0, if $\mathsf{Vf}_\Xi(\pi_o, \mathsf{ctx}) = 0$, where ctx is as above.
3. Output 1.

Self-Inspection. In order to prove, or dis-prove, ownership of a signature of the form $((e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad})$, the user proves that it knows a valid signature from the issuer on its public key, and that its pseudonym for the given scope is either equal, or non-equal, to the pseudonym in question. That is, $\mathsf{SOpn}(\mathsf{ssk}, \mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{ipk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}), m)$ performs the following steps:

1. Output $\perp$, if $\mathsf{Vf}_{\Omega}(\mathsf{ipk}, \mathsf{opk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, m) = 0$.
2. Set $b = 1$, if $\mathsf{Gen}_{\Theta}(\mathsf{usk}, h) = \mathsf{nym}_{m\|\mathsf{pad}}$ and $b = 0$ otherwise.
3. Compute the following zero-knowledge proof of knowledge:

$$\pi_u \xleftarrow{\$} \mathsf{ZKP}\Big[(\mathsf{usk}, \mathsf{upk}, \mathsf{ssk}) : \mathsf{upk} = \mathsf{Gen}_{\Theta}(\mathsf{usk}, \mathtt{setup}) \wedge$$
$$\mathsf{nym}_{m\|\mathsf{pad}} \sim \mathsf{Gen}_{\Theta}(\mathsf{usk}, h) \ \wedge \ \mathsf{Vf}_{\Omega}(\mathsf{ipk}, \mathsf{ssk}, (\mathsf{upk}, \mathsf{opk})) = 1\Big](\mathsf{ctx}),$$

   where $\sim \in \{=, \neq\}$, depending on whether ownership is proven $(b = 1)$ or denied $(b = 0)$, where

$$\mathsf{ctx} = (\mathsf{pp}_{\Omega}, m, \mathsf{opk}, \mathsf{ipk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, \sim).$$

4. It outputs $\pi_{\mathsf{signer}} \leftarrow \pi_u$.

To verify whether the user with user public key upk is really (not) accountable for a signature σ, algorithm $\mathsf{SJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_u, \mathsf{upk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, m, b)$ behaves as follows:

1. Output 0, if $\mathsf{Vf}_{\Omega}(\mathsf{ipk}, \mathsf{opk}, (e, h, \pi_s, \mathsf{nym}_{m\|\mathsf{pad}}), \mathsf{pad}, m) = 0$.
2. Output 0, if $\mathsf{Vf}_{\Xi}(\pi_u, \mathsf{ctx}) = 0$, where ctx is as above.
3. Output 1.

Signature-Linking. To prove whether two signatures were issued by the same signer, $\mathsf{Lnk}(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, (e_0, h_0, \pi_{s,0}, \mathsf{nym}_{m\|\mathsf{pad},0}), (e_1, h_1, \pi_{s,1}, \mathsf{nym}_{m\|\mathsf{pad},1}))$ checks whether or not the two ciphertexts decrypt to the same value. That is, the algorithm performs the following steps:

1. Check the validity of $\pi_{s,0}$ and $\pi_{s,1}$. If one is not valid, return $\perp$.
2. Compute $\widehat{\mathsf{upk}}_i \leftarrow \mathsf{Dec}(\mathsf{osk}, e_i)$ for $i = 0, 1$.
3. Set $b = 0$, if $\widehat{\mathsf{upk}}_0 \neq \widehat{\mathsf{upk}}_1$ and $b = 1$ otherwise.
4. Compute the following zero-knowledge proof of knowledge:

$$\pi_l \xleftarrow{\$} \mathsf{ZKP}\Big[(\mathsf{osk}) : \mathsf{OKVf}(\mathsf{osk}, \mathsf{opk}) = 1 \wedge$$
$$\mathsf{Dec}(\mathsf{osk}, e_0) \sim \mathsf{Dec}(\mathsf{osk}, e_1)\Big](\mathsf{ctx}),$$

   where $\sim \in \{=, \neq\}$, depending on whether equality of the signers is proven $(b = 1)$ or not $(b = 0)$, and

$$\mathsf{ctx} = (\mathsf{pp}_{\Omega}, (e_0, h_0, \pi_{s,0}, \mathsf{nym}_{m\|\mathsf{pad},0}),$$
$$(e_1, h_1, \pi_{s,1}, \mathsf{nym}_{m\|\mathsf{pad},1}), \mathsf{opk}, b).$$

5. It outputs $\pi_{\mathsf{link}} \leftarrow \pi_l$

To verify whether two signatures were indeed (not) issued by the same signer, $\mathsf{LnkJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_l, (e_0, h_0, \pi_{s,0}, \mathsf{nym}_{m\|\mathsf{pad},0}), (e_1, h_1, \pi_{s,1}, \mathsf{nym}_{m\|\mathsf{pad},1}), \mathsf{pad}_0, \mathsf{pad}_1, m_0, m_1, b)$ behaves as follows:

1. Output 0, if, for some $i \in \{0, 1\}$, $\mathsf{Vf}_{\Omega}(\mathsf{ipk}, \mathsf{opk}, (e_i, h_i, \pi_{s,i}, \mathsf{nym}_{m\|\mathsf{pad},i}), \mathsf{pad}_i, m_i) = 0$.
2. Output 0, if $\mathsf{Vf}_{\Xi}(\pi_l, \mathsf{ctx}) = 0$, where ctx is as above.
3. Output 1.

Self-Linking of Signatures. A user can use $\mathsf{SLnk}(\mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{ipk}, (e_0, h_0, \pi_{s,0}, \mathsf{nym}_{m\|\mathsf{pad},0}), (e_1, h_1, \pi_{s,1}, \mathsf{nym}_{m\|\mathsf{pad},1}), \mathsf{pad}_0, \mathsf{pad}_1, m_0, m_1)$ to show that the same signer is or is not accountable for the given signatures, as long as the proving user was at least the signer of one of the signatures. To do so, the algorithm behaves as follows:

1. Let $b_i = 1$, if $\mathsf{nym}_{m_i\|\mathsf{pad}_i} = \mathsf{Gen}_\Theta(\mathsf{usk}, h_i)$, and $b_i = 0$ otherwise for $i = 0, 1$, and $b = 1$ if $b_0 = b_1 = 1$ and $b = 0$ otherwise, where $h_i = \mathcal{H}(m_i\|\mathsf{pad}_i)$.
2. Output $\perp$, if $b_0 = b_1 = 0$.
3. Compute the following zero-knowledge proof of knowledge:

$$\pi_l \xleftarrow{\$} \mathsf{ZKP}\Big[(\mathsf{usk}, \mathsf{upk}, \iota) : \mathsf{nym}_{m_\iota\|\mathsf{pad}_\iota} = \mathsf{Gen}_\Theta(\mathsf{usk}, h_\iota) \wedge \\ \mathsf{nym}_{m_{1-\iota}\|\mathsf{pad}_{1-\iota}} \sim \mathsf{Gen}_\Theta(\mathsf{usk}, h_{1-\iota})\Big](\mathsf{ctx}),$$

where $\sim \in \{=, \neq\}$, depending on whether equality of the signers is proven ($b = 1$) or not ($b = 0$), $h_i = \mathcal{H}(m_i\|\mathsf{pad}_i)$, and

$$\mathsf{ctx} = (\mathsf{pp}_\Omega, \mathsf{opk}, b, (e_0, h_0, \pi_{s,0}, \mathsf{nym}_{m\|\mathsf{pad},0}), \\ (e_1, h_1, \pi_{s,1}, \mathsf{nym}_{m\|\mathsf{pad},1}), \mathsf{pad}_0, \mathsf{pad}_1, m_0, m_1).$$

4. It outputs $\pi_{\mathsf{linku}} \leftarrow \pi_l$

To verify a proof whether two signatures were indeed (or not resp.) issued by the same signer, algorithm $\mathsf{SLnkJdg}(\mathsf{opk}, \mathsf{ipk}, \pi_l, (e_0, h_0, \pi_{s,0}, \mathsf{nym}_{m\|\mathsf{pad},0}), (e_1, h_1, \pi_{s,1}, \mathsf{nym}_{m\|\mathsf{pad},1}), \mathsf{pad}_0, \mathsf{pad}_1), m_0, m_1, b)$ behaves as follows:

1. Output 0, if $\mathsf{Vf}_\Omega(\mathsf{ipk}, \mathsf{opk}, (e_i, h_i, \pi_{s,i}, \mathsf{nym}_{m\|\mathsf{pad},i}), \mathsf{pad}_i, m_i) = 0$ for $i = 0$ or $i = 1$.
2. Output 0, if $\mathsf{Vf}_\Xi(\pi_l, \mathsf{ctx}) = 0$, where ctx is as above.
3. Output 1.

The full proof of the following theorem is given in Appendix B.

Theorem 1. *If Π is an IND-CPA secure encryption scheme, Σ is an unforgeable signature scheme,* ZKP *is a weakly simulation-sound extractable zero-knowledge proof system, and Θ is a collision-resistant unlinkable scope-exclusive pseudonym system, then the above construction yields a secure group signature scheme Ω.*

Proof (Sketch). Correctness follows by inspection. Non-frameability follows from soundness of the used Ξ and - somewhat surprisingly - the unlinkability of Θ. Anonymity follows from the ZK-property of Ξ, the IND-CPA security of Π, randomly drawn paddings and the unlinkability of Θ. Traceability follows from the soundness of Ξ and the unforgeability of the signature scheme. Trace-Soundness follows from the soundness of Ξ and the collision-resistance of Θ. Finally, opening-privacy follows from the randomly drawn paddings.

5 Implementation and Performance Evaluation

To demonstrate the efficiency and practicability of the our construction presented above, we provide performance benchmarks for a concrete instantiation based on ElGamal encryption [21], Abe et al.'s structure preserving signature scheme [2], the scope-exclusive pseudonym system by Camenisch et al. [15], and zero-knowledge proofs of knowledge based on the Schnorr protocol (Σ-protocols) [35] and the Fiat-Shamir heuristic [20]. Concrete details on the construction can be found in Appendix C.

The implementation uses IAIK's ECCelerate pairings library[6]. In particular, the underlying pairing curves are "SNARK_2", while the used hardware was a rather old PC with an Intel Corei5-2400 running at 3.1Ghz, and 16GiB of RAM. No performance optimizations were implemented, and only a single thread does the computations, while the random oracle is implemented as SHA-512. An overview over the results, based on 1'000 runs, is given in Figure 7, Figure 8, Table 1, and Table 2. Here, $\mathsf{Opn}_\perp$ means that the opener opens a signature in a standard way, while Opn_0 means that an opener denies a signature. Note, verifying a proof that a signature is denied does not require to verify the signature itself.

[6] https://jce.iaik.tugraz.at/sic/Products/Core_Crypto_Toolkits/ECCelerate

Table 1: Performance Measurements in ms

	Sign_{Ω}	Vf_{Ω}	SOpn	SJdg	Lnk	LnkJdg	SLnk	$\mathsf{SLnkJdg}$
Min.:	75	103	165	206	211	210	215	213
1/4:	76	106	168	209	214	214	218	217
Med.:	77	107	169	210	216	215	219	218
3/4:	77	108	171	212	217	217	221	220
9/10:	78	109	173	215	220	219	224	222
19/20:	79	110	176	219	222	221	228	225
Max.:	92	129	197	249	252	252	262	257
Avg.:	72	107	170	211	216	216	221	219
SD:	1.67	2.37	3.24	4.22	3.91	4.32	4.55	4.24

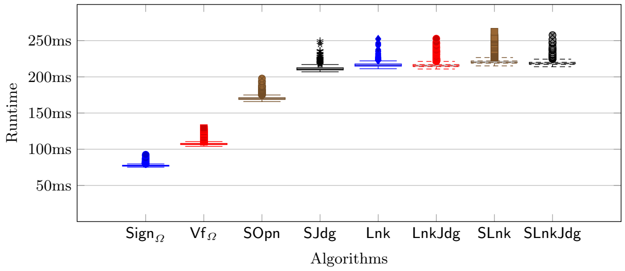


Fig. 7: Performance Evaluation Results

Table 2: Additional Performance Measurements in ms

	GKG	OKG	UKG	Join	$\mathsf{Opn}_{\perp}$	$\mathsf{Jdg}_{1,\perp}$	Opn_0	Jdg_0
Min.:	8	1	1	75	106	107	110	109
1/4:	8	1	1	77	108	109	112	111
Med.:	8	1	1	77	109	110	113	112
3/4:	8	1	1	78	110	111	114	113
9/10:	8	1	1	79	111	112	115	114
19/20:	9	1	1	81	113	113	117	116
Max.:	10	1	2	91	130	138	131	131
Avg.:	8	1	1	78	109	110	113	113
SD:	0.31	0.06	0.11	1.78	2.37	3.04	2.53	2.50

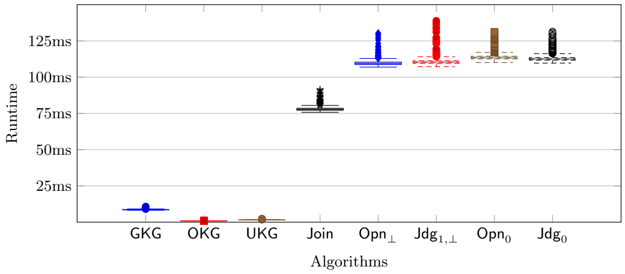


Fig. 8: Additional Performance Evaluation Results

In this overview, we focus on the, in our opinion, most interesting algorithms. Namely, parameter generation is omitted, at this is only a one-time setup, while the non-equal algorithms perform the same amount of work as their counterparts.

Still, our construction performs each operation well below a second, and thus can be considered truly practical. The increased running times for the judges are easily explained, as they also have to verify signatures. Moreover, compared to other implementations [33], our non-optimized implementation can be considered competitive.

6 Conclusion

We have introduced practical group-signatures with privacy-friendly openings. Our notion allows the opener not only to fully open signatures, but to also prove that a given signatures does not stem from a particular user, and — the same time — to link signatures without revealing the user which created those signatures. Moreover, the opener no longer requires the message in question to open a signature. Our framework grants the same possibilities to the users. Combined, these capabilities lessen the requirements on the opener, and increase the appl0icability of group signatures. Our construction, which is exclusively based on standard primitives, is practical.

References

1. M. Abe, S. S. M. Chow, K. Haralambiev, and M. Ohkubo. Double-trapdoor anonymous tags for traceable signatures. *Int. J. Inf. Sec.*, 12(1):19–31, 2013.
2. M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups. In *CRYPTO*, 2011.
3. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In *CRYPTO*, pages 255–270, 2000.
4. F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *EuroS&P*, pages 301–315, 2017.
5. E. Bangerter. *Efficient Zero Knowledge Proofs of Knowledge for Homomorphisms.* PhD thesis, Ruhr University Bochum, 2005.
6. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *EUROCRYPT*, pages 614–629, 2003.
7. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.
8. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, pages 136–153, 2005.
9. D. Bernhard, O. Pereira, and B. Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *ASIACRYPT*, pages 626–643, 2012.
10. P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi. Get shorty via group signatures without encryption. In *SCN*, pages 381–398, 2010.
11. O. Blazy, D. Derler, D. Slamanig, and R. Spreitzer. Non-interactive plaintext (in-)equality proofs and group signatures with verifiable controllable linkability. In *CT-RSA*, pages 127–143, 2016.
12. O. Blazy and D. Pointcheval. Traceable signature with stepping capabilities. In *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, pages 108–131, 2012.
13. J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth. Foundations of Fully Dynamic Group Signatures. In *ACNS*, pages 117–136, 2016.
14. E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *CCS*, pages 132–145, 2004.
15. J. Camenisch, S. Krenn, A. Lehmann, G. L. Mikkelsen, G. Neven, and M. Ø. Pedersen. Formal treatment of privacy-enhancing credential systems. In *SAC*, pages 3–24, 2015.
16. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Crypto*, pages 61–76, 2002.
17. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *Crypto*, pages 410–424, 1997.
18. D. Chaum and E. van Heyst. Group Signatures. In *EUROCRYPT*, pages 257–265, 1991.
19. S. Faust, M. Kohlweiss, G. Azzurra Marson, and D. Venturi. On the non-malleability of the fiat-shamir transform. In *IndoCrypt*, pages 60–79, 2012.
20. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, pages 186–194, 1986.
21. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO*, pages 10–18, 1984.
22. H. Ge and S. R. Tate. Traceable signature: Better efficiency and beyond. In *ICCSA, Part III*, pages 327–337, 2006.
23. J. Y. Hwang, A. Lee, B.-H. Chung, H. S. Cho, and D. Nyang. Group signatures with controllable linkability for dynamic membership. *Inf. Sci.*, 222:761–778, 2013.
24. A. Ishida, K. Emura, G. Hanaoka, Y. Sakai, and K. Tanaka. Group Signature with Deniability: How to Disavow a Signature. In *CANS*, pages 228–244, 2016.
25. A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *Eurocrypt*, pages 571–589, 2004.

26. A. Kiayias and M. Yung. Secure scalable group signature with dynamic joins and separable authorities. *IJSN*, 1(1/2):24–45, 2006.
27. S. Krenn. *Bringing Zero-Knowledge Proofs of Knowledge to Practice.* PhD thesis, University of Fribourg, 2012.
28. B. Libert, S. Ling, K. Nguyen, and H. Wang. Zero-Knowledge Arguments for Lattice-Based Accumulators: Logarithmic-Size Ring Signatures and Group Signatures Without Trapdoors. In *EUROCRYPT II*, pages 1–31, 2016.
29. B. Libert and M. Yung. Efficient traceable signatures in the standard model. *Theor. Comput. Sci.*, 412(12-14):1220–1242, 2011.
30. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *SAC*, pages 184–199, 1999.
31. M. Manulis, N. Fleischhacker, F. Günther, F. Kiefer, and B. Poettering. Group signatures: Authentication with privacy. Technical report, TU Darmstadt, 2012.
32. M. Manulis, A.-R. Sadeghi, and J. Schwenk. Linkable democratic group signatures. In *ISPEC*, pages 187–201, 2006.
33. K. Potzmader, J. Winter, D. M. Hein, C. Hanser, P. Teufl, and L. Chen. Group signatures on mobile devices: Practical experiences. In *TRUST*, pages 47–64, 2013.
34. Y. Sakai, J. C. N. Schuldt, K. Emura, G. Hanaoka, and K. Ohta. On the security of dynamic group signatures: Preventing signature hijacking. In *PKC*, pages 715–732, 2012.
35. C.-P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *CRYPTO*, pages 239–252, 1989.
36. D. Slamanig, R. Spreitzer, and T. Unterluggauer. Adding controllable linkability to pairing-based group signatures for free. In *ISC*, pages 388–400, 2014.
37. D. Slamanig, R. Spreitzer, and T. Unterluggauer. Linking-based revocation for group signatures: A pragmatic approach for efficient revocation checks. In *Mycrypt*, pages 364–388, 2016.
38. D. X. Song. Practical forward secure group signature schemes. In *CCS*, pages 225–234, 2001.
39. Q. Wu, W. Susilo, Y. Mu, and F. Zhang. Ad hoc group signatures. In *IWSEC*, pages 120–135, 2006.

A Additional Preliminaries

A.1 Non-Interactive Zero-Knowledge Proof of Knowledge Systems

Let L be an NP-language with associated witness relation R, i.e., such that $L = \{x \mid \exists w : R(x,w) = 1\}$. In a nutshell, a zero-knowledge non-interactive proof of knowledge (a.k.a. signature proof of knowledge) allows to verify that the generator of that proofs knows a witness w for some statement x without revealing that witness. We use the definitions by Faust et al. [19].

We only consider Σ-protocols:

Definition 11 (Σ-Protocol). *A Σ-protocol $\Sigma = (\mathcal{P}, \mathcal{V})$ for a language L is a three-round public-coin IPS where $\mathcal{P}$ and $\mathcal{V}$ are PPT algorithms which also provide soundness, honest-verifier zero-knowledge (HVZK), completeness and special soundness, where $\mathcal{P}$ moves first.*

A complete transcript of such a protocol run is denoted as $\langle \mathcal{P}(x,w), \mathcal{V}(x) \rangle$. To make this protocol non-interactive, the message send from the verifier to the prover ("the commitment") is generated by making a random-oracle call consisting of the first message sent by $\mathcal{P}$ and the statement to be proven, i.e., the Fiat-Shamir (FS) transform. We use the following notation for brevity, tailored for FS. Note, all parties have explicit access to a random oracle $\mathcal{H}$.

Definition 12 (ZKPs). *A zero-knowledge non-interactive proof of knowledge system* ZKP *consists of two algorithms* $\{\mathsf{Prv}_\Xi, \mathsf{Vf}_\Xi\}$*, such that:*

Prv_Ξ**.** *This algorithm outputs the proof π, on input of the statement x to be proven, and the corresponding witness w using the FS-transform: $\pi \xleftarrow{\$} \mathsf{Prv}_\Xi(x,w)$.*

Vf_Ξ**.** *This algorithm verifies the proof π, w.r.t. to some statement x, where $d \in \{0,1\}$: $d \leftarrow \mathsf{Vf}_\Xi(x,\pi)$.*

For brevity, the Camenisch-Stadler notation [17] is used to express the statements proven in non-interactive, weakly simulation-sound extractable, zero-knowledge. In more detail, the notation $\pi \xleftarrow{\$} \mathsf{ZKP}\,[(w) : R(x,w) = 1]$ denotes the computation of a non-interactive, weakly simulation-sound extractable, zero-knowledge proof of knowledge, where all values not in the parentheses are assumed to be public. For example, let L be defined by the following NP-relation:

$$((g,h,y,z),(a,b)) \in R \iff y = g^a \quad \wedge \quad z = g^b h^a$$

Hence, $\pi \xleftarrow{\$} \mathsf{ZKP}\left[(a,b) : y = g^a \wedge z = g^b h^a\right]$ denotes a corresponding non-interactive proof-of-knowledge (PoK) of witness (a,b) with respect to the statement (g,h,y,z), for the above language L, while sometimes only "verify π" is used for verification. It is assumed that the public parameters, and the statement to be proven, are also input to the proof system, and public. This is not make explicit to increase readability.

Security. We now explicitly define zero-knowledge and weak simulation-sound extractability.

Weak Simulation-Sound Extractability. This security notion says that an adversary cannot generate a proof π^* for a statement it does not know a witness for, while the proof-system is also of knowledge, i.e., the witness w can be extracted from any non-simulated proof π, if the extractor SIM can rewind the adversary. Clearly, this also implies that the proof-system is non-malleable.

Definition 13 (Weak Simulation-Sound Extractability). *Let L be a language in NP. Consider a proof system* ZKP *(where $\mathcal{H}$ is a random oracle) for L with a ZK-simulator* $\mathsf{SIM} = (\mathsf{SIM}_1, \mathsf{SIM}_2)$ *(sharing state). Let* SIM_1 *simulate the random oracle, while* SIM_2 *calls the HVZK-simulator and programs the random-oracle accordingly to make proofs (even "proofs" of false statements) verify. We say that* ZKP *is weakly simulation-sound extractable with extraction error μ w.r.t.* SIM *in the programmable random-oracle model, if for all PPT adversaries $\mathcal{A}$ there exists an efficient algorithm* SIM_3 *with access to all transcripts such that the following holds. Let*

$$\begin{aligned} \mathsf{acc} &= \Pr[(x^*, \pi^*) \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{SIM}_1(\cdot), \mathsf{SIM}_2(\cdot)}(\rho) : (x^*, \pi^*) \notin \mathcal{T}; \mathsf{Vf}_{\Xi}(x^*, \pi^*) = 1] \\ \mathsf{ext} &= \Pr[(x^*, \pi^*) \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{SIM}_1(\cdot), \mathsf{SIM}_2(\cdot)}(\rho) : \\ &\quad w^* \overset{\$}{\leftarrow} \mathsf{SIM}_3(x^*, \pi^*; \rho, \mathcal{T}_{\mathcal{H}}, \mathcal{T} : (x^*, \pi^*) \notin \mathcal{T}; (x^*, w^*) \in R_L] \end{aligned}$$

where ρ is the adversary's random tape, $\mathcal{T}_{\mathcal{H}}$ the random oracle table and $\mathcal{T}$ the answers by SIM_2*. Then, there exists a constant $d > 0$ and a polynomial p such that whenever $\mathsf{acc} \geq \mu$, we have $\mathsf{ext} \geq \frac{1}{p}(\mathsf{acc} - \mu)^d$.*

Note, however, that this probability can be made exponentially close to 1 by standard repetition techniques. We can thus safely assume that extraction is possible with overwhelming probability.

Zero-Knowledge. In a nutshell, zero-knowledge says that the receiver of the proof π does not learn anything except the validity of the statement.

Definition 14 (Zero-Knowledge). *A non-interactive proof system* ZKP *is said to be zero-knowledge, if for a fixed language L, for any efficient adversary $\mathcal{A}$, there exists an efficient simulator* SIM *such that there exists a negligible function ν such that:*

$$\left| \Pr[\mathsf{Zero\text{-}Knowledge}^{\mathsf{Prv}_{\Xi}}_{\mathcal{A},\mathsf{SIM},L}(\lambda) = 1] - \frac{1}{2} \right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 9.

Faust et al. have already shown that the FS-transform yields such a ZK-system [19].

```
Experiment $\mathsf{Zero\text{-}Knowledge}^{\mathsf{ZKP}}_{\mathcal{A},\mathsf{SIM},L}(\lambda)$
  $b \overset{\$}{\leftarrow} \{0,1\}$
  $a \overset{\$}{\leftarrow} \mathcal{A}^{P_b(\cdot,\cdot),\mathcal{H}(\cdot)}(1^\lambda)$
    where oracle $P_0$ on input $(x, w)$:
      if $R(x, w) = 1$, return $\pi \overset{\$}{\leftarrow} \mathsf{Prv}_{\Xi}(x, w)$
      return $\perp$
    and oracle $P_1$ on input $(x, w)$:
      if $R(x, w) = 1$, return $\pi \overset{\$}{\leftarrow} \mathsf{SIM}(x)$
      return $\perp$
  return 1, if $a = b$
  return 0
```

Fig. 9: ZKP Zero-Knowledge

```
Experiment $\mathsf{eUNF\text{-}CMA}_{\mathcal{A}}^{\Sigma}(\lambda)$
  $\mathsf{pp}_{\mathsf{SYS}} \xleftarrow{\$} \mathsf{PGen}_{\mathsf{SYS}}(1^{\lambda})$
  $\mathsf{pp}_{\Sigma} \xleftarrow{\$} \mathsf{PGen}_{\Sigma}(\mathsf{pp}_{\mathsf{SYS}})$
  $(\mathsf{sk}_{\Sigma}, \mathsf{pk}_{\Sigma}) \leftarrow \mathsf{KG}_{\Sigma}(\mathsf{pp}_{\mathsf{SYS}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sig}'_{\Sigma}(\mathsf{sk}_{\Sigma}, \cdot)}(\mathsf{pk}_{\Sigma})$
    where oracle $\mathsf{Sig}'_{\Sigma}$ on input $m$:
      let $\sigma \leftarrow \mathsf{Sig}_{\Sigma}(\mathsf{sk}_{\Sigma}, m)$
      set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
      return $\sigma$
  return 0, if $m^* \in \mathcal{Q}$
  return 1, if $\mathsf{Vf}_{\Sigma}(\mathsf{pk}_{\Sigma}, m^*, \sigma^*) = 1$
  return 0
```

Fig. 10: Σ Unforgeability

A.2 Digital Signatures

Definition 15 (Digital Signatures Σ). *A signature scheme Σ is a tuple $\{\mathsf{PGen}_{\Sigma}, \mathsf{KG}_{\Sigma}, \mathsf{Sig}_{\Sigma}, \mathsf{Vf}_{\Sigma}\}$ of PPT algorithms such that:*

PGen_{Σ}**.** *This algorithm generates the public parameter of the scheme: $\mathsf{pp}_{\Sigma} \xleftarrow{\$} \mathsf{PGen}_{\Sigma}(\mathsf{pp}_{\mathsf{SYS}})$.*
KG_{Σ}**.** *This algorithm outputs the public and corresponding private key:*

$$(\mathsf{sk}_{\Sigma}, \mathsf{pk}_{\Sigma}) \xleftarrow{\$} \mathsf{KG}_{\Sigma}(\mathsf{pp}_{\Sigma})$$

Sig_{Σ}**.** *This algorithm gets as input sk_{Σ}, the message $m \in \mathcal{M}$, and outputs a signature:*

$$\sigma \xleftarrow{\$} \mathsf{Sig}_{\Sigma}(\mathsf{sk}_{\Sigma}, m)$$

Vf_{Σ}**.** *This deterministic algorithm receives as input a public key pk_{Σ}, a message m and a signature σ and outputs a decision bit $d \in \{0, 1\}$:*

$$d \leftarrow \mathsf{Vf}_{\Sigma}(\mathsf{pk}_{\Sigma}, m, \sigma)$$

Correctness. We now define correctness.

Definition 16 (Correctness.). *A digital signature scheme Σ is correct, if for all*

Security. Besides completeness, a signature scheme Σ need to satisfy *eUNF-CMA security.* In a nutshell, we require that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with a valid signature σ^* for a new message m^*. Moreover, the adversary $\mathcal{A}$ can adaptively query for new signatures.

Definition 17 (Unforgeability). *A signature scheme Σ is* unforgeable*, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathsf{eUNF\text{-}CMA}_{\mathcal{A}}^{\Sigma}(1^{\lambda}) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Fig. 10.

A.3 Public-Key Encryption Schemes

Definition 18 (Public-Key Encryption Schemes). *A public-key encryption scheme Π consists of four algorithms $\{\mathsf{PGen}_{\Pi}, \mathsf{KG}_{\Pi}, \mathsf{Enc}_{\mathsf{Enc}}, \mathsf{Dec}_{\mathsf{Enc}}, \mathsf{KVf}_{\Pi}\}$, such that:*

```
Experiment $\mathsf{IND\text{-}CPA}_{\mathcal{A}}^{\Pi}(\lambda)$:
  $\mathsf{pp}_{\mathsf{SYS}} \xleftarrow{\$} \mathsf{PGen}_{\mathsf{SYS}}(1^{\lambda})$
  $\mathsf{pp}_{\Pi} \xleftarrow{\$} \mathsf{PGen}_{\Pi}(\mathsf{pp}_{\mathsf{SYS}})$
  $(\mathsf{sk}_{\Pi}, \mathsf{pk}_{\Pi}) \xleftarrow{\$} \mathsf{KG}_{\Pi}(\mathsf{pp}_{\Pi})$
  $b \xleftarrow{\$} \{0,1\}$
  $((m_0^*, m_1^*), state_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_{\Pi})$
  If $m_0^* \notin \mathcal{M} \vee m_1 \notin \mathcal{M}$:
    let $c^* \leftarrow \perp$
  Else:
    let $c^* \xleftarrow{\$} \mathsf{Enc}_{\mathsf{Enc}}(\mathsf{pk}_{\Pi}, m_b^*)$
  $a \xleftarrow{\$} \mathcal{A}(state_{\mathcal{A}}, c^*)$
  return 1, if $a = b$
  return 0
```

Fig. 11: Π IND-CPA Security

PGen_{Π}. *This algorithm outputs the public parameters of the scheme:*

$$\mathsf{pp}_{\Pi} \xleftarrow{\$} \mathsf{PGen}_{\Pi}(\mathsf{pp}_{\mathsf{SYS}})$$

It is assumed that pp_{Π} is implicit input to all other algorithms.

KG_{Π}. *This algorithm outputs the public and private key, on input pp_{Π}:*

$$(\mathsf{sk}_{\Pi}, \mathsf{pk}_{\Pi}) \xleftarrow{\$} \mathsf{KG}_{\Pi}(\mathsf{pp}_{\Pi})$$

$\mathsf{Enc}_{\mathsf{Enc}}$. *This algorithm gets as input the public key pk_{Π}, and the message $m \in \mathcal{M}$ to encrypt. It outputs a ciphertext:*

$$c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{Enc}}(\mathsf{pk}_{\Pi}, m)$$

$\mathsf{Dec}_{\mathsf{Enc}}$. *This algorithm outputs a message m (or $\perp$, if the ciphertext is invalid) on input sk_{Π}, and a ciphertext c:*

$$m \leftarrow \mathsf{Dec}_{\mathsf{Enc}}(\mathsf{sk}_{\Pi}, c)$$

KVf_{Π}. *This algorithm decides whether a given public key pk_{Π} corresponds to a given secret key sk_{Π},*

$$d \leftarrow \mathsf{KVf}_{\Pi}(\mathsf{pk}_{\Pi}, \mathsf{sk}_{\Pi})$$

where $d \in \{0,1\}$.

Security. We require that the encryption scheme Π is IND-CPA secure.

Definition 19 (IND-CPA Security). *An encryption scheme Π is IND-CPA secure, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathsf{IND\text{-}CPA}_{\mathcal{A}}^{\mathsf{ENC}}(\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 11.

B Proof of Theorem 1

Each property is proven on its own.

Correctness. This property follows by inspection.

Non-Frameability. We now prove that our scheme is non-frameable:

Game 0: The original non-frameability game.

Game 1: As Game 0, but the ZK-proofs for signing, linking and opening are simulated using SIM.

Transition - Game 0 → Game 1: An adversary $\mathcal{A}$ distinguishing this replacement can be used to break the ZK-property of the proof-system. The reduction works as follows. Our reduction $\mathcal{B}$ receives oracle access to a prove-oracle. All other values are generated honestly. For every proof generated, it passes the statement to be proven (note, the statements to be proven are still legit) to the proof-oracle. The result is embedded into the response for $\mathcal{A}$. So, if $\mathcal{A}$ notices a difference, so does $\mathcal{B}$ with the same probability. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$ follows.[7]

Game 2: As Game 1, but we abort, if we cannot extract a valid witness w for a verifying proof which was not simulated.

Transition - Game 1 → Game 2: An adversary $\mathcal{A}$ outputting such a proof π^* can be used to break the weak simulation-sound extractability property of the proof-system. The reduction works as follows. It receives ρ and embeds it as $\mathcal{A}$'s random coins. It then rewires calls to the random-oracle $\mathcal{T}_{\mathcal{H}}$ to SIM_1 and uses SIM_2 to simulate all proofs generated. Then, whenever the adversary outputs π^*, which cannot be extracted using SIM_3, we break the weak simulation-sound extractability property of the proof-system, which, by definition, cannot happen.[8] $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{wsse}}(\lambda)$ follows.

Game 3: As Game 2, but we abort, if the adversary is able to generate a message/signature pair which points to the challenge upk, but was never generated by the honest user. Note, this case also happens if the adversary queries such a signature to the opening and linking oracles. Moreover, in this case the opening information τ is *not* considered part of the signature σ.

Transition - Game 2 → Game 3: This can be used to break the unlinkability of the pseudonym-system. The reduction $\mathcal{B}$ itself is rather simple: It first receives the public parameters pp_{Θ}. It embeds the public parameters accordingly, i.e., all other parameters are generated as in the prior hop, while joining can be done honestly. It calls its own challenge oracle $\mathcal{O}_0$ to receive a pseudonym (on the correct message and padding, and also for setup) which it can embed into the response for each signature to be generated (and public key during joining). The self-opening oracles for signatures generated by the Sign'_{Ω}-oracle are fully simulated, pointing to the signer. For signatures not coming from that oracle, they deny ownership (but are simulated).

We stress that noticing such a forged signatures is simple: $\mathcal{B}$ queries its pseudonym-oracles to receive two $\mathsf{nym}'_{m\|\mathsf{pad}}$ on the received $m\|\mathsf{pad}$. If neither $\mathsf{nym}'_{m\|\mathsf{pad}}$ matches the one in the signature not generated by the Sign'_{Ω}-oracle, a forgery has happened, already meeting the winning requirements.

The self-linking oracles are simulated in the same fashion: if both signatures come from Sign'_{Ω}-oracle, the proof is fully simulated, stating that the signatures are linkable. In the case that only one signature comes from the Sign'_{Ω}-oracle, but they should be linkable, the winning conditions are met and thus this case does not require any attention. If both signatures are not generated by the Sign'_{Ω}-oracle, $\mathcal{B}$ returns $\perp$. Again, the case that at least one signature should make the self-linking oracle output a proof already meets the winning conditions.

Finally, after the adversary outputs a forgery (or, as described above, puts one into the oracles), $\mathcal{B}$ extracts usk (by using the canonical extractor SIM_3). $\mathcal{B}$ can then trivially break unlinkability of the pseudonym system by recalculating pseudonyms. In particular, the challenge scope can be chosen randomly from the set of non-seen scopes, and then simply verified using the extracted usk. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nym-unlink}}(\lambda)$ follows.

This proves that our scheme is non-frameable, as the adversary has no other way to win the game.

Anonymity. We now prove that our scheme is anonymous:

Game 0: The original anonymity game in the case $b = 0$.

Game 1: As Game 0, but the ZK-proofs for signing, linking and opening are simulated using SIM.

Transition - Game 0 → Game 1: An adversary $\mathcal{A}$ distinguishing this replacement can be used to break the ZK-property of the proof-system. The reduction works as follows. Our reduction $\mathcal{B}$ receives oracle access to a prove-oracle. All other values are generated honestly. For every proof generated, it passes the statement to be proven (note, the statements to be proven are still legit) to the proof-oracle. The result is embedded into the response for $\mathcal{A}$. So, if $\mathcal{A}$ notices a difference, so does $\mathcal{B}$ with the same probability. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$ follows.[9]

[7] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

[8] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

[9] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

Game 2: As Game 1, but we abort, if we cannot extract a valid witness w for a verifying proof which was not simulated.

Transition - Game 1 → Game 2: An adversary $\mathcal{A}$ outputting such a proof π^* can be used to break the weak simulation-sound extractability property of the proof-system. The reduction works as follows. It receives ρ and embeds it as $\mathcal{A}$'s random coins. It then rewires calls to the random-oracle $\mathcal{T}_{\mathcal{H}}$ to SIM_1 and uses SIM_2 to simulate all proofs generated. Then, whenever the adversary outputs π^*, which cannot be extracted using SIM_3, we break the weak simulation-sound extractability property of the proof-system, which, by definition, cannot happen.[10] $|\Pr[S_1]-\Pr[S_2]| \leq \nu_{\mathsf{wsse}}(\lambda)$ follows.

Game 3: As Game 2, but we abort if the adversary was able to query a valid signature to one of the opening or linkings oracles, pointing to one of the honest users, which has never been generated by one of the honest users. Note, the information τ is only required at the self-opening and self-linking oracles, but are *not* part of the signature.

Transition - Game 2 → Game 3: An adversary $\mathcal{A}$ generating such a valid message/signature pair can be turned into an adversary $\mathcal{B}$ against the unlinkability definition of the underlying pseudonym-system. In particular, the reduction $\mathcal{B}$ works as follows. $\mathcal{B}$ receives pp_{Θ}, embedding it honestly. Likewise, it generates opk and osk honestly. It also honestly generates two user key-pairs $(\mathsf{usk}', \mathsf{upk}')$ and $(\mathsf{usk}, \mathsf{upk})$. $\mathcal{B}$ then gives $(\mathsf{opk}, \mathsf{upk}, \mathsf{upk}')$ to $\mathcal{A}$. The oracles are simulated as follows. Joining can be done honestly. If a signature is to be generated, i.e., a call to Sign'_{Ω} it can be generated honestly, but the pseudonyms are generated using $\mathcal{O}_0$ for upk and $\mathcal{O}_1$ for upk'. Joining is done in the same fashion, but with the special scope, using the same oracles. Opn' is simulated as follows. If the signature was generated using LoRSig, $\perp$ is returned (if ipk matches). In all other cases, the answer is returned honestly (if the signature points to one of the users, but was never generated by them, the winning condition is already met; See below). The same is true for Lnk', but quantified over any signature input. Likewise, calls to SOpn' are performed honestly, except for simulated signatures; here, the proofs need to be simulated, pointing to the correct user. This is also true for the SLnk'-oracle. So far, the simulation is perfect. However, when the adversary makes a query to the opening or linking-oracles for which the opener would return a proof which makes one of the users accountable, while they are not, $\mathcal{B}$ can extract a secret key usk. The reduction $\mathcal{B}$ can then use this key to recalculate pseudonyms (the challenge can be obtained on a random pseudonym not yet seen), directly breaking the unlinkability of the used pseudonym system. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{unlink}}(\lambda)$ follows.

Game 4: As Game 3, but we replace each encryption e with an encryption of 0 for the challenge opk for the signature generation.

Transition - Game 3 → Game 4: An adversary distinguishing this replacement can be used to break the IND-CPA security of the used encryption scheme using a standard hybrid argument. Let q_s be an upper bound on the number of signatures generated w.r.t. the challenge opk. Further, let Game 4.0 be the same as Game 3. In Game 4.i we replace the content of the first i encryptions with a 0, i.e., in Game 4.q_s all encryption encrypt a 0. Let $\mathcal{A}$ be an adversary which can distinguish this replacement. We can then construct a reduction $\mathcal{B}$ which breaks the IND-CPA security of the underlying encryption scheme. The reduction $\mathcal{B}$ works as follows. It receives pk_{Π} as its own challenge and pp_{Π}. Both are embedded into the values the adversary receives — all other values are generated as in the prior hop. For all queries up to ith one, the content of the encryption is replaced with a 0. On the ith query, however, $\mathcal{B}$ asks its own challenger with either the correct value or a 0. The result is embedded into the response. All oracles are still answered as in the prior game, with one notable exception: if a ciphertext is to be decrypted during opening (which is not necessary for simulated ones, as the values are known by $\mathcal{B}$), $\mathcal{B}$ uses SIM_3 to obtain the plaintext. Then, whatever is output by $\mathcal{A}$, is also output by $\mathcal{B}$. $|\Pr[S_3] - \Pr[S_4]| \leq q_s \nu_{\mathsf{IND-CPA}}(\lambda)$ follows, where q_s is the number of signatures generated.

Game 5: As Game 4, but we abort, if the same padding was drawn twice for any honest user in the system.

Transition - Game 4 → Game 5: As the pads are drawn completely randomly from $\{0,1\}^{2\lambda}$, the probability is negligible, as it is bound by the birthday paradox. $|\Pr[S_4] - \Pr[S_5]| \leq q_s^2/2^{2\lambda}$ follows, where q_s is the number of signatures generated.

Game 6: As Game 5, but switch to $b = 1$.

Transition - Game 5 → Game 6: Clearly, as now everything is independent from the input values (but usk for the pseudonyms), the only option left is that the adversary can link pseudonyms. The reduction $\mathcal{B}$ proceeds as follows. It draws a random bit $b' \xleftarrow{\$} \{0,1\}$. It receives pp_{Θ} and embeds it accordingly. All other values are generated as in the prior game. Joining can be done honestly. Then, $\mathcal{B}$ calls its own oracles $\mathcal{O}_{b'}$ (for upk^0) and $\mathcal{O}_{1-b'}$ (for upk^1) to receive the pseudonyms for scope `setup` and then embeds them into the public keys. Likewise, each pseudonym for signing is calculated using the oracles provided. Note, proofs are already simulated, and thus the secret key is not

[10] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

required — how signatures should be opened/linked is directly obvious from the transcript of the signing oracles, as we already excluded forged signature (regardless of τ). $\mathcal{B}$ then chooses an index i from $\{1, 2, \ldots, q_s\}$, where q_s is the number of signatures generated by the LoR-oracle, and calls the challenge oracle with the correct scope and embeds the result in the signature. For all calls afterwards, $\mathcal{B}$ continues using its oracles $\mathcal{O}_{b'}$ (for upk^0) and $\mathcal{O}_{1-b'}$ (for upk^1) as before. Then, whatever $\mathcal{A}$ outputs, is also output by the reduction. Conditioned on the probability that $b' = b$ (which happens in exactly 50% of the cases), the simulation is perfect. In the case $b' \neq b$ there is simulation glitch, as $\mathcal{B}$ embeds the wrong pseudonym. Thus, we obtain $|\Pr[S_5] - \Pr[S_6]| \leq 2\nu_{\mathsf{nym-unlink}}(\lambda)$.

This proves that our scheme is anonymous.

Traceability. We now prove that our scheme is traceable:

Game 0: The original traceability game.
Game 1: As Game 0, but the ZK-proofs for signing, linking and opening are simulated using SIM.
Transition - Game 0 → Game 1: An adversary $\mathcal{A}$ distinguishing this replacement can be used to break the ZK-property of the proof-system. The reduction works as follows. Our reduction $\mathcal{B}$ receives oracle access to a prove-oracle. All other values are generated honestly. For every proof generated, it passes the statement to be proven (note, the statements to be proven are still legit) to the proof-oracle. The result is embedded into the response for $\mathcal{A}$. So, if $\mathcal{A}$ notices a difference, so does $\mathcal{B}$ with the same probability. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$ follows.[11]
Game 2: As Game 1, but we abort, if we cannot extract a valid witness w for a verifying proof which was not simulated.
Transition - Game 1 → Game 2: An adversary $\mathcal{A}$ outputting such a proof π^* can be used to break the weak simulation-sound extractability property of the proof-system. The reduction works as follows. It receives ρ and embeds it as $\mathcal{A}$'s random coins. It then rewires calls to the random-oracle $\mathcal{T}_{\mathcal{H}}$ to SIM_1 and uses SIM_2 to simulate all proofs generated. Then, whenever the adversary outputs π^*, which cannot be extracted using SIM_3, we break the weak simulation-sound extractability property of the proof-system, which, by definition, cannot happen.[12] $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{wsse}}(\lambda)$ follows.
Game 3: As Game 2, but we abort if the adversary was able to generate a signature for a member not joined (note, osk is known and thus this condition can easily be checked).
Transition - Game 2 → Game 3: As always the knowledge of sig is proven, the reduction can extract it using the canonical extractor SIM_3. The reduction receives the public key to forge, and embeds it into ipk (the parameters are embedded as well). Signatures given to the adversary can be generated using the signature oracle provided. Moreover, as the user never joined, the signature is fresh, as all other proofs are simulated. Thus, sig can be extracted and breaks the unforgeability of the underlying signature scheme. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{unf-cma}}(\lambda)$ follows.
Game 4: As Game 3, but we abort if the adversary was able to generate a signature for which the accountable party cannot be determined or linking does not work.
Transition - Game 3 → Game 4: This means that the proof contained in the signature is bogus. The statement and proof can trivially be extracted from the values given. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{zkwsimsound}}(\lambda)$ follows.

This proves that our scheme is traceable, as the adversary has no other way to win.

Trace-Soundness. We now prove that our scheme is trace-sound:

Game 0: The original trace-soundness game.
Game 1: As Game 0, but we abort, if we cannot extract a valid witness w for a verifying proof which was not simulated.
Transition - Game 0 → Game 1: An adversary $\mathcal{A}$ outputting such a proof π^* can be used to break the weak simulation-sound extractability property of the proof-system. The reduction works as follows. It receives ρ and embeds it as $\mathcal{A}$'s random coins. It then rewires calls to the random-oracle $\mathcal{T}_{\mathcal{H}}$ to SIM_1 and uses SIM_2 to simulate all proofs generated. Then, whenever the adversary outputs π^*, which cannot be extracted using SIM_3, we break the weak simulation-sound extractability property of the proof-system, which, by definition, cannot happen.[13] $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{wsse}}(\lambda)$ follows.

[11] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

[12] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

[13] In a formal sense, we need to make this hop for every language L involved. However, this is a pure technicality and no additional insight would be given. We therefore make this no longer explicit.

Game 2: We now abort, if the conditions of the trace-soundness game are met.

Transition - Game 1 → Game 2: We are now in two cases. In the first case, we have colliding pseudonyms with different secret keys. The secret keys can be extracted from the proofs using SIM_3. The reduction $\mathcal{B}$ works as follows. It receives pp_Θ from its own challenger. It embeds it accordingly. All other values are generated as in the prior game. Then, whenever the above case happens, $\mathcal{B}$ can return $(\mathsf{usk}_0, \mathsf{usk}_1, \mathsf{sc})$.

In the other case we have two proofs π_0^*, and π_1^*, proving different statements (one of which must be wrong), trivially breaking the soundness of the zero-knowledge proof system. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nymcoll}} + 3\nu_{\mathsf{zkwsimsound}}(\lambda)$ follows, as $\mathcal{B}$ can only return one proof and do not know which one is bogus.

This proves that our scheme is trace-sound, as the adversary has no other way to win the game.

Opening-Privacy. We now prove that our scheme is opening-private:

Game 0: The original opening-privacy game.

Game 1: As Game 0, but the ZK-proofs for signing and opening are simulated.

Transition - Game 0 → Game 1: An adversary distinguishing this replacement can clearly be used to break the ZK-property of the proof-system. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$ follows.

Game 2: As Game 1, but we abort if the adversary was able to query a valid signature to one of the opening oracles, pointing to one of the honest users, which has never been generated by one of the honest users.

Transition - Game 1 → Game 2: An adversary $\mathcal{A}$ generating such a valid message/signature pair can trivially be turned into an adversary $\mathcal{B}$ against the non-frameability requirement. In particular, the reduction $\mathcal{B}$ works as follows. $\mathcal{B}$ receives upk. $\mathcal{B}$ then gives upk to $\mathcal{A}$. The oracles are simply rewired to $\mathcal{B}$'s own challenger. So far, the simulation is perfect. However, when the adversary makes a query to the opening-oracles for which the opener would return a proof which makes upk accountable, while it is not, $\mathcal{B}$ can return that signature to its own challenger. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{frame}}(\lambda)$ follows.

Game 3: As Game 2, but each h is no longer calculated using the message and the padding in the LoR-oracle, but using a random value $h \xleftarrow{\$} \{0,1\}^{2\lambda}$, while we abort, if the adversary makes a query p which results in h.

Transition - Game 2 → Game 3: As $\mathcal{H}$ maps values to $\{0,1\}^{2\lambda}$, this only happens with negligible probability, i.e., $|\Pr[S_1] - \Pr[S_2]| \leq \frac{q_h q_s}{2^{2\lambda}}$, where q_h is the number of random oracle queries and q_s the number of signatures generated. Note, all oracles can thus be simulated honestly.

This proves that our scheme is opening-private, as the signatures are now completely independent from the message.

C Concrete Instantiation

Parameter Generation. On input 1^λ, PGen_Ω outputs $\mathsf{pp}_{\mathsf{SYS}} = (1^\lambda, \mathsf{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, where q is a prime of sufficient length, and $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric pairing, where the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$.

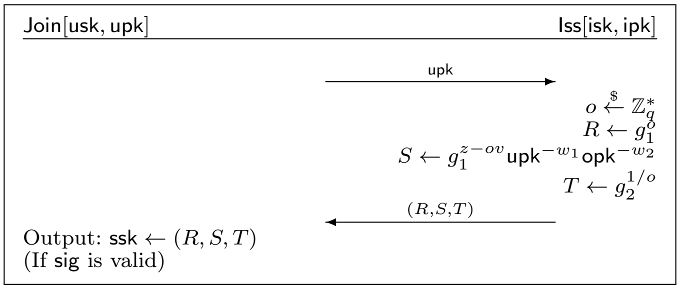
The parameters pp_Σ generated by $\mathsf{PGen}_\Sigma(\mathsf{pp}_{\mathsf{SYS}})$ consist of generators g_i of $\mathbb{G}_i$ for $i = 1, 2$. No additional parameters pp_Θ generated by $\mathsf{PGen}_\Theta(\mathsf{pp}_{\mathsf{SYS}})$ are needed; however, the pseudonym system has access to a random oracle $\mathcal{O}_\Theta$ mapping arbitrary bitstrings to elements of $\mathbb{G}_1$. The parameters pp_Π generated by $\mathsf{PGen}_\Pi(\mathsf{pp}_{\mathsf{SYS}})$ consist of a generator g of $\mathbb{G}_1$ (potentially, $g = g_1$). Finally, the zero-knowledge building block again has access to a random oracle $\mathcal{O}_\Xi$ mapping arbitrary bitstrings to elements of $\mathbb{Z}_q$.

Key Generation. The issuer generates its key pair using the key generation algorithm of the Abe et al. signature scheme [2]. That is, it chooses $v, w_1, w_2, z \xleftarrow{\$} \mathbb{Z}_q$ and defines $(V, W_1, W_2, Z) = (g_2^v, g_2^{w_1}, g_2^{w_2}, g_2^z)$. Finally, the GKG outputs $(\mathsf{isk}, \mathsf{ipk}) \leftarrow ((v, w_1, w_2, z), (V, W_1, W_2, Z))$.

To generate its key pair, the opener's key generation algorithm OKG draws $\mathsf{osk} \xleftarrow{\$} \mathbb{Z}_q$ and defines $\mathsf{opk} \leftarrow g^{\mathsf{osk}}$.

The user's key generation algorithm UKG draws $\mathsf{usk} \xleftarrow{\$} \mathbb{Z}_q$ and defines $\mathsf{upk} \leftarrow \mathcal{O}_\Theta(\texttt{setup})^{\mathsf{usk}}$ as a scope-exclusive pseudonym to the special scope $\texttt{setup}$.

Join / Issuance. As described in the generic construction, the interactive protocol $\langle\mathsf{Join}, \mathsf{Iss}\rangle$ simply consists in the user transferring its pseudonym to the issuer, who then signs this value and opk using its own key material. It then sends back the signature to the user. The concrete operations are depicted in Protocol 2.


Join[usk, upk]
Iss[isk, ipk]
upk
$o \xleftarrow{\$} \mathbb{Z}_q^*$
$R \leftarrow g_1^o$
$S \leftarrow g_1^{z-ov}\mathsf{upk}^{-w_1}\mathsf{opk}^{-w_2}$
$T \leftarrow g_2^{1/o}$
(R,S,T)
Output: $\mathsf{ssk} \leftarrow (R,S,T)$
(If sig is valid)

Prot. 2: Concrete instantiation of the protocol $\langle\mathsf{Join};\mathsf{Iss}\rangle$.

Sign / Verify. In order to sign a message m, a user computes an ElGamal encryption of its upk, computes a fresh pseudonym for the given message-padding-pair. It then generates a signature of knowledge on the given context, proving that the above computations were done correctly and that upk was originally signed by the issuer. Note that the consistency of the secret user keys used for nym and upk – and thereby also that the signer is the valid owner of upk – is implicitly demonstrated by showing that the plaintext of the ElGamal encryption corresponds to $\mathcal{O}_\Theta(\mathtt{setup})^{\mathsf{usk}}$.

More precisely, the signer executes the following steps (note that the third step is a purely technical work that is needed to make the proof goal in the next step compatible with standard Schnorr-like protocols, cf., e.g., Bangerter [5]):

1. $s \xleftarrow{\$} \mathbb{Z}_q$, $e \leftarrow (g^s, \mathsf{upk} \cdot \mathsf{opk}^s)$
2. $\mathsf{nym} \leftarrow \mathcal{H}(m\|\mathsf{pad})^{\mathsf{usk}}$
3. $k \xleftarrow{\$} \mathbb{Z}_q^*$, $\hat{T} \leftarrow T^{-k}$
4. $$\pi \xleftarrow{\$} \mathsf{ZKP}\Big[\Big(\mathsf{usk}, s, R, S, k\Big) : e = (g^s, \mathcal{O}_\Theta(\mathtt{setup})^{\mathsf{usk}}\mathsf{opk}^{\mathsf{s}}) \wedge \\ \mathsf{e}(R,V)\mathsf{e}(S,g_2)\mathsf{e}(\mathcal{O}_\Theta(\mathtt{setup})^{\mathsf{usk}}, W_1) = \mathsf{e}(g_1, Z)\mathsf{e}(\mathsf{opk}, W_2)^{-1} \wedge \\ \mathsf{nym} = \mathcal{O}_\Theta(h)^{\mathsf{usk}} \ \wedge\ \mathsf{e}(R,\hat{T})\mathsf{e}(g_1,g_2)^k = 1\Big](h, \mathsf{ctx}),$$
   where $h = \mathcal{H}(m\|\mathsf{pad})$.
5. Output: $(\sigma, \tau) \leftarrow (e, \mathsf{nym}, (\hat{T}, \pi), \mathsf{pad})$

In order to verify a signature, Vf_Ω behaves as described in Section 4.

Open / Judge. In order to open a signature σ on message m, the opener behaves as follows if $\mathsf{upk}' = \bot$, i.e., if the signature should indeed be opened and no (in)equality to a target user key is to be proven. It first checks the validity of π. It decrypts the ciphertext e to obtain the user's public key $\hat{\mathsf{upk}} \leftarrow e_2 e_1^{-\mathsf{osk}}$ and generates a zero-knowledge proofs that it did so correctly:

$$\pi_{\mathsf{opener}} \xleftarrow{\$} \mathsf{ZKP}\Big[(\mathsf{osk}) : \mathsf{opk} = g^{\mathsf{osk}} \ \wedge\ \hat{\mathsf{upk}} e_2^{-1} = e_1^{-\mathsf{osk}}\Big](\mathsf{ctx}).$$

To verify the validity of the opener's claim, Jdg first checks whether $\mathsf{Vf}_\Omega(\mathsf{opk}, \mathsf{ipk}, \sigma, \tau, m) = 1$, and outputs 0 otherwise. It then outputs whatever Vf_Ξ outputs on input π_{opener} for the given context.

In the case that $\mathsf{upk}' \neq \bot$, the algorithm computes b as in the generic construction. If $b = 1$, it computes the very same proof as above, using upk' instead of $\hat{\mathsf{upk}}$. Otherwise, if $b = 0$ and the opener should therefore prove that the target upk' is not accountable for the signature, the following steps are performed:

1. The algorithm draws a blinding factor $v \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $E \leftarrow (\mathsf{upk}' e_2)^v$, $D_1 \leftarrow e_1^v$, and $D_2 \leftarrow D_1^{-\mathsf{osk}}$.
2. It computes the following proof:
   $$\pi'_{\mathsf{opener}} \xleftarrow{\$} \mathsf{ZKP}\Big[(\mathsf{osk}, v) : \mathsf{opk} = g^{\mathsf{osk}} \ \wedge\ E = (\mathsf{upk}' e_2)^v \ \wedge \\ D_1 = e_1^v \ \wedge\ D_2 = D_1^{-\mathsf{osk}}\Big](\mathsf{ctx}),$$
   where ctx also includes D_1, D_2, E.

3. It sets $\pi_{\mathsf{opener}} \leftarrow (\pi'_{\mathsf{opener}}, D_1, D_2, E)$.

Upon verification, the algorithm Jdg in addition to verifying π'_{opener} also checks that $D_2 \neq E$ and $D_1, D_2, E \neq 1$.

Note that from this last proof it follows that $E \cdot D_2^{-1} = (\mathsf{upk}' e_2^{-1})^v (e_1^{-v \cdot \mathsf{osk}})^{-1} = \mathsf{upk}'^v (e_1^{osk} e_2^{-1})^v = (\mathsf{upk}' \hat{\mathsf{upk}}^{-1})^v$, where $\hat{\mathsf{upk}}$ denotes the decryption (e_1, e_2) under the opener's key. By checking $E \neq D_2$, it thus follows that the correct decryption is different from the target user public key upk'. Note further that the real user's identity remains computationally hidden under DLOG, which however does not weaken the user's privacy, as the encryption (e_1, e_2) of $\hat{\mathsf{upk}}$ itself already depends on the same assumption anyways, and therefore no additional overhead to achieve perfect instead of computational zero-knowledge in this construction is necessary.

Self-Inspection. To self-inspect a valid signature $(\sigma, \tau) = (e, \mathsf{nym}, (\hat{T}, \pi), \mathsf{pad})$, the user's algorithm SJdg (dis-)proves that the pseudonym in question was generated by the user by showing that the user is the legitimate owner of the public key $\mathsf{upk} = \mathcal{O}_\Theta(\mathtt{setup})^{\mathsf{usk}}$, and the pseudonym for the respective $m \| \mathsf{pad}$ is (un-)equal to nym.

Specifically, the algorithm computes the following proof π_u to prove ownership of a signature (i.e., in the case that $b = 1$):

1. $k \xleftarrow{\$} \mathbb{Z}_q^*$, $\hat{T}' \leftarrow T^{-k}$
2. $$\begin{aligned}\pi'_u \xleftarrow{\$} \mathsf{ZKP}\Big[\Big(\mathsf{usk}, R, S, k\Big) : \mathsf{nym} = \mathcal{O}_\Theta(h)^{\mathsf{usk}} \ \wedge \ \mathsf{upk} = \mathcal{O}_\Theta(\mathtt{setup})^{\mathsf{usk}} \ \wedge \\ e(R, V)\mathsf{e}(S, g_2) = \mathsf{e}(g_1, Z)\mathsf{e}(\mathsf{opk}, W_2)^{-1}\mathsf{e}(\mathsf{upk}, W_1)^{-1} \ \wedge \\ \mathsf{e}(R, \hat{T}')\mathsf{e}(g_1, g_2)^k = 1\Big](\mathsf{ctx}),\end{aligned}$$
   where $h = \mathcal{H}(m \| \mathsf{pad})$.
3. and outputs $\pi_u \leftarrow (\hat{T}, \pi'_u)$.

To deny a signature (i.e., if $b = 0$), the algorithm behaves as follows:

1. $k \xleftarrow{\$} \mathbb{Z}_q^*$, $\hat{T}' \leftarrow T^{-k}$
2. $r \xleftarrow{\$} \mathbb{Z}_q^*$ and $D \leftarrow \mathsf{nym}^r \mathcal{O}_\Theta(h)^{-\mathsf{usk} \cdot r}$
3. $$\begin{aligned}\pi'_u \xleftarrow{\$} \mathsf{ZKP}\Big[\Big(\mathsf{usk}, R, S, k, r\Big) : D = \mathsf{nym}^r \mathcal{O}_\Theta(h)^{-\mathsf{usk} \cdot r} \ \wedge \\ \mathsf{e}(R, V)\mathsf{e}(S, g_2)\mathsf{e}(\mathcal{O}_\Theta(\mathtt{setup})^{\mathsf{usk}}, W_1) = \mathsf{e}(g_1, Z)\mathsf{e}(\mathsf{opk}, W_2)^{-1} \ \wedge \\ \mathsf{e}(R, \hat{T}')\mathsf{e}(g_1, g_2)^k = 1\Big](\mathsf{ctx}),\end{aligned}$$
   where the multiplication is resolved using standard techniques found in the literature, e.g., Krenn [27], and $h = \mathcal{H}(m \| \mathsf{pad})$.
4. and outputs $\pi_u \leftarrow (D, \hat{T}, \pi'_u)$.

To verify the correctness of a user's claim, SJdg first checks whether $\mathsf{Vf}_\Omega(\mathsf{opk}, \mathsf{ipk}, \sigma, \tau, m) = 1$, and outputs 0 otherwise. In case of a denied signature, it also checks that $D \neq 0$ and outputs 0 otherwise. It then outputs whatever Vf_Ξ outputs on input π_{signer} for the given context.

Signature-Linking. On input $(\mathsf{osk}, \mathsf{opk}, \mathsf{ipk}, (e_i, \mathsf{nym}_i, (\hat{T}_i, \pi_i)))$, where for $i = 0, 1$, algorithm Lnk first checks the validity of each π_i, computes $\hat{\mathsf{upk}}_i = e_{i,1} e_{i,2}^{-\mathsf{osk}}$ for $i = 0, 1$. It then defines $b = 1$ if $\hat{\mathsf{upk}}_0 = \hat{\mathsf{upk}}_1$ and $b = 0$ otherwise.

If $b = 1$, it then computes the following zero knowledge proof of knowledge:

$$\pi_l = \mathsf{ZKP}\left[(\mathsf{osk}) : \mathsf{opk} = g^{\mathsf{osk}} \wedge e_{1,1} \cdot e_{2,1}^{-1} = \left(e_{1,2} \cdot e_{2,2}^{-1}\right)^{\mathsf{osk}}\right](\mathsf{ctx}).$$

Verification follows straightforward from the black-box construction.

Otherwise, if $b = 0$, the algorithm follows the same logics as Opn to show that $\hat{\mathsf{upk}}_0 \neq \hat{\mathsf{upk}}_1$. The algorithm draws $v \xleftarrow{\$} \mathbb{Z}_q^*$, and sets $D_1 \leftarrow \left(e_{1,2} \cdot e_{2,2}^{-1}\right)^v$, $D_2 \leftarrow D_1^{\mathsf{osk}}$, and $E \leftarrow \left(e_{1,1} \cdot e_{2,1}^{-1}\right)^v$. It then computes:

$$\pi_l' = \mathsf{ZKP}\Big[(\mathsf{osk}, v) : \mathsf{opk} = g^{\mathsf{osk}} \ \wedge \ D_1 = \left(e_{1,2} \cdot e_{2,2}^{-1}\right)^v \ \wedge \\ D_2 = D_1^{\mathsf{osk}} \ \wedge \ E = \left(e_{1,1} \cdot e_{2,1}^{-1}\right)^v \Big](\mathsf{ctx})\,,$$

and sets $\pi_l = (\pi_l', E, D_1, D_2)$, and adds (E, D_1, D_2) to ctx. Upon verification of π_l, the verifier not only verifies π_l', but also checks that $D_2 \neq E$.

Self-Linking of Signatures. After having checked whether or not the same signer is accountable for both signatures, the algorithm behaves as follows if $b = 1$. It computes:

$$\pi_l \xleftarrow{\$} \mathsf{ZKP}\left[(\mathsf{usk}) : \bigwedge_{i=0}^{1} \mathsf{nym}_{m_i \| \mathsf{pad}_i} = \mathcal{O}_\Theta(h_i)^{\mathsf{usk}}\right](\mathsf{ctx})\,.$$

where $h_i = \mathcal{H}(m_i \| \mathsf{pad}_i)$. In the case that $b = 0$ it computes:

$$\pi_l \xleftarrow{\$} \mathsf{ZKP}\Big[(\mathsf{usk}) : \bigvee_{i=0}^{1} \big(\mathsf{nym}_{m_i \| \mathsf{pad}_i} = \mathcal{O}_\Theta(h_i)^{\mathsf{usk}} \ \wedge \\ \mathsf{nym}_{m_{1-i} \| \mathsf{pad}_{1-i}} \neq \mathcal{O}_\Theta(h_{i-1})^{\mathsf{usk}}\big)\Big](\mathsf{ctx})\,,$$

where the inequality is resolved as in the previous algorithms and $h_i = \mathcal{H}(m_i \| \mathsf{pad}_i)$.