Group Signatures with User-Controlled and Sequential Linkability

Jesus Diaz[1] and Anja Lehmann[2]

[1] IBM Research – Zurich, Rüschlikon, Switzerland
jdv@zurich.ibm.com
[2] Hasso-Plattner-Institute, University of Potsdam, Germany
anja.lehmann@hpi.de

Abstract. Group signatures allow users to create signatures on behalf of a group while remaining anonymous. Such signatures are a powerful tool to realize privacy-preserving data collections, where e.g., sensors, wearables or vehicles can upload authenticated measurements into a data lake. The anonymity protects the user's privacy yet enables basic data processing of the uploaded unlinkable information. For many applications, full anonymity is often neither desired nor useful though, and selected parts of the data must eventually be correlated after being uploaded. Current solutions of group signatures do not provide such functionality in a satisfactory way: they either rely on a trusted party to perform opening or linking of signatures, which clearly conflicts with the core privacy goal of group signatures; or require the user to decide upon the linkability of signatures *before* they are generated.

In this paper we propose a new variant of group signatures that provides linkability in a flexible and user-centric manner. Users – and only they – can decide before and after signature creation whether they should remain linkable or be correlated. To prevent attacks where a user omits certain signatures when a *sequence* of events in a certain section (e.g., time frame), should be linked, we further extend this new primitive to allow for sequential link proofs. Such proofs guarantee that the provided sequence of data is not only originating from the same signer, but also occurred in that exact order and contains *all* of the user's signatures within the time frame. We formally define the desired security and privacy properties, propose a provably secure construction based on DL-related assumptions and report on a prototypical implementation of our scheme.

1 Introduction

Group signatures [17,5] extend conventional signatures to protect the signers' identity. Signers remain anonymous within the anonymity set defined by the members of a group formed by users who request to join and are accepted by the manager. Anyone with the group public key can verify signatures. To avoid abusing anonymity, an *opener* can usually re-identify the signer of any signature. This enables accountability and further processing if data needs to be more identifiable or linked, but requires full trust on the opener to ensure privacy.

Schemes with trusted openers. To reduce this dependency, alternatives quickly sprouted. In group signatures with Verifier Local Revocation, verifiers can keep local lists of *revoked* signers, not requiring them to open incoming signatures [10]. Traceable signatures [23,18] add an extra trusted entity who, after opening a signature by any given member, can produce member-specific trapdoors that can be used to link signatures originating by them. Convertably linkable signatures remove the opener, but incorporate a party who can (non-transitively) blindly link signatures within sets of queried signatures [22]. Recently, also blind variants for central opening have been proposed [25]. Still, all these alternatives use some sort of central entity for opening or linking, which needs to be fully trusted to ensure privacy. While this trust can be distributed [13], this still gives control to a set of central entities rather than users.

Schemes with user-controlled linkability. Instead of relying on trusted parties, it may suffice to let signers control which signatures will be linkable, and when. This is also ideal from a privacy perspective, as users retain full control. In this vein, Direct Anonymous Attestation (DAA) [6,12] and anonymous credential systems [15], also aimed at preserving signer/holder privacy, follow this approach. They enable user-controlled linkability through deterministically computed pseudonyms (from a scope and the user's key) within each signature. This makes all signatures for the same scope automatically linkable. Otherwise, they remain unlinkable. Such *implicit linking* has the drawback of being static: a signature that was decided to be unlinkable to some or all other signatures, will remain unlinkable forever. Thus, use cases with even a remote probability of needing to link signatures a posteriori would require to make them all linkable by default, eliminating all privacy.

Further, relying on the more privacy-friendly option of user-controlled and implicit linkability instead of having an almighty opener, makes formally defining the desired security and privacy properties of such group signatures much more challenging. In fact, to date no satisfactory security model for DAA in the form of accessible game-based security notions is known; we refer to [12,6] for a summary of the long line of failed security notions in that respect.

Alternatively, some existing group signatures offer user-controlled a posteriori linking or opening of previously anonymous signatures: In [28] users can claim signatures by outputting their secret key which allows to test whether a signature stemmed from that user. But this is an all-or-nothing approach, immediately destroying privacy of all the user's signatures and thus is unsuitable for most realistic scenarios. The recent work by Krenn et al. [25] implement a more flexible *explicit linking* by enabling users to issue link proofs for two (or, in theory, more) signatures. However, their model still crucially relies on the presence of a trusted opener to model and prove the desired security properties. Thus, even if only explicit linking would be needed, the scheme must allow full opening through a central entity in order to fit their model and hope for any provable security guarantees.

Ideally, one would hope for group signatures supporting both implicit and explicit linking to increase utility and, for scenarios handling sensitive data, without trusted parties that can unilaterally remove privacy.

1.1 Our Contributions

In this paper we provide the first provably secure group signatures that are purely user-centric, i.e., where only the user can control the linkage of her signatures. To allow for the necessary flexibility, our solution supports both implicit and explicit linkability. That is, the user can make signatures linkable with respect to pseudonyms when she generates them, and also link signatures with different pseudonyms afterwards through explicit link proofs.

Security model without opener, and for implicit and explicit linking. Our first challenge was to provide meaningful security notions when no opener is available that can be leveraged, e.g., to express who is a valid member of the group. Instead, we take inspiration from security models for DAA [6,12] to express membership of groups through linking. We define *anonymity* by requiring that it must not be possible to link signatures by the same user, except when she decides to make them linkable by default, or when she explicitly links them. For *traceability*, (1) it must not be possible to create signatures that are not traceable to any valid member of the group, and (2) it must not be possible to explicitly link signatures originating from different (possibly corrupt) users. Finally, for *non-frameability* we require that (1) no signature can be implicitly linkable to another honest signature unless it was honestly generated by the same user – who also made both signatures linkable by default, and (2) no adversary can explicitly link honest and dishonest signatures, or honest signatures that have not been explicitly linked by their signer. Note that we give two variants for both traceability and non-frameability. This is needed due to the possibility to implicitly and explicitly link signatures, and is a direct consequence of leveraging linkability to replace the opener. We emphasize that, to the best of our knowledge, implicit linking has not been modelled previously for group signatures – let alone in combination with explicit linking.

Sequential link proofs. When the pseudonymous signatures are over data with inherent order properties – e.g., time series – just re-establishing linkage is not enough. Therein, it may be needed to attest that the linked messages are given in the same order in which they were produced, and without omitting (possibly relevant) ones. For instance, smart vehicles in Intelligent Transportation Systems (ITSs) are required to send measurements to a data lake. There, the order of a sequence of events may be useful to detect anomalies: e.g., a vehicle reporting 35-45-30-40 litres of fuel in a short timespan is probably an anomaly, while one reporting 45-40-35-30 is probably not. Or, again, in contact tracing systems, where pseudonyms are reused during a limited time, after which new ones are derived. Users may eventually be required to reveal their pseudonymous data spanning several of those pseudonyms, and omitting specific chunks of this data (or altering the order) may preclude effective contact tracing. In these use cases, the number of pseudonymously signed messages that may be required to be linked can be expected to be of at least many tens (and possibly a few hundreds) of signatures, in short time spans. Additionally, order may be relevant in less throughput-demanding scenarios. For instance, it may have very different implications when a person fails to pay X mortgage fees in a row, than the case when the X defaults correspond to months very distant in time.

This motivates our next contribution. We extend our previous model and construction to enable sequential link proofs: signers can prove that a sequence of signatures was produced in the specified order, and no signature is being omitted. To model this, we introduce a new unforgeability property, *sequentiality*, ensuring that honest-then-corrupt users cannot create sequential proofs for wrongly ordered sequences, nor omitting signatures. Our extended construction builds on efficient hash-chain ideas from anonymous payment systems [26].

Efficient construction with batch proofs for linking. We give an efficient construction realizing our model. Pseudonymous signatures are computed using the scope-exclusive nym approach from DAA and anonymous credentials, where the pseudonym is deterministically derived from a scope and the same secret key in the user's credential. This gives implicit linkage. For explicitly linking signatures, we propose a new way to batch the signatures being linked, leveraging the fact that pseudonyms are group elements that can be "aggregated". This leads to an efficient mechanism for linking large sets of signatures.

Implementation and comparison. To further assess efficiency of our constructions, we implement them and report on the obtained experimental results (check Appendix A for notes on the implementation and a demo). Both the basic scheme and sequential extension outperform the most related previous work [25]: we link sets of ∼100 signatures in ∼40ms, while [25] requires ∼300ms for linking only 2 signatures (besides requiring a trusted opener.)

2 Preliminaries

Notation. $\mathbb{G} = \langle g \rangle$ denotes a cyclic group $\mathbb{G}$ generated by g, $a \leftarrow A(\cdot)$ denotes a obtained by applying algorithm A, $a \leftarrow_\$ S$ means a is picked uniformly from set S, and $[n]$ denotes the closed interval $[1, n]$. H and H' are cryptographic hash functions. Signed messages are represented as a tuple of elements. When arguing about sets of such tuples, $\boldsymbol{\Sigma}$ denotes a set, and Σ_i the i-th element in $\boldsymbol{\Sigma}$. $\boldsymbol{\Sigma}_o$ is an ordered set, and $A_o \in_o S_o$ denotes that A_o appears in S_o, respecting order.

Bilinear maps. Let $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle, \mathbb{G}_T$ be three cyclic groups of prime order p, where an efficient mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ exists. e satisfies bilinearity, i.e., $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$; non-degeneracy, i.e., $e(g_1, g_2)$ generates $\mathbb{G}_T$; and efficiency, i.e., there exists $\mathsf{PG}(1^\tau)$ efficiently generating bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ as above, and computing $e(a, b)$ is efficient for any $a \in$

$\mathbb{G}_1, b \in \mathbb{G}_2$. Moreover, we use Type-III bilinear maps [21], i.e., $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no efficiently computable homomorphisms between them.

Hardness assumptions. We base the security of our scheme in the well known Discrete Logarithm and DDH assumptions [16] and in the q-SDH assumption for Type-III pairings [9], which we informally recall next.

q-SDH assumption (for Type-III pairings [9].) Given $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $\chi \in \mathbb{Z}_p$, and a $(\mathbb{G}_1^{q+1}, \mathbb{G}_2^2)$ tuple $(g_1, g_1^{\chi}, g_1^{(\chi^2)}, ..., g_1^{(\chi^q)}, g_2, g_2^{\chi})$, it is computationally unfeasible for any polynomial-time machine to output a tuple $(g_1^{\frac{1}{x+\chi}}, x) \in \mathbb{G}_1 \times \mathbb{Z}_p \setminus \{-\chi\}$.

BBS+ signatures and Pseudonyms. We rely on the BBS+ signature scheme proposed in [1] for Type-II pairings, and Type-III pairings in [11].

We use the following convention for BBS+ operations, for some previously generated Type-III pairing group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$:

- **Key Generation.** Compute $(h_1, h_2) \leftarrow_\$ \mathbb{G}_1^2$, $y \leftarrow_\$ \mathbb{Z}_p^*$, $W \leftarrow_\$ g_2^y$. Set $sk \leftarrow y$ and $pk \leftarrow (W, h_1, h_2)$.
- **Signing.** Given a message m (assumed to be in $\mathbb{Z}_p$, pick $x, s \leftarrow_\$ \mathbb{Z}_p^*$ and compute $A \leftarrow (g_1 h_1^s h_2^m)^{\frac{1}{x+y}}$. The signature is the tuple (A, x, s).
- **Verification.** Given a signature (A, x, s) over a message m, supposedly from $pk = (W, h_1, h_2)$, check that $e(A, W g_2^x) = e(g_1 h_1^s h_2^m, g_2)$.

We extend the proof of knowledge in BBS+ signatures to prove correctness of the pseudonyms that signers generate.

For pseudonyms, we follow [14]. Roughly, with the help of a hash function, pseudonyms are deterministically generated from a scope *scp* and a private key *sk* as $\mathsf{H}(scp)^{sk}$.

Proof protocols. We use non-interactive proofs of knowledge obtained through the Fiat-Shamir transform [20]. $\mathsf{SPK}\{(x, r) : h = h_1^x h_2^r\}(ctx, m)$, denotes a signature of knowledge of (x, r) meeting the condition to the right of the colon, for public message m, and parameters ctx to prevent malleability attacks [7]. For verification, we write $\mathsf{SPKVerify}(\pi, ctx, m)$, returning 1 (correct) or 0 (incorrect).

Additional building blocks. We rely on an append-only bulletin board BB and pseudo random functions (PRFs). PRFs generate pseudorandom output from a secret key and arbitrary inputs. $\mathsf{PRF.KeyGen}(1^\tau) \rightarrow k$ generates the keys, and $\mathsf{PRF.Eval}(k, m) \rightarrow r$ pseudorandomness r from key k and message m. The BB is assumed to verify the data before writing, and written data cannot be erased.

3 Scheme with User-Controlled Linkability (UCL)

In this section we present our basic group signature scheme with user-controlled and selective linkability. We start by presenting the general syntax, then describe how the desired security properties can be formulated without the presence of an opening entity, and finally present our secure instantiation.

The core contribution of this section is the new security model that captures the desired security and privacy properties without a central (trusted) entity and allows for selective, user-centric linkability. The proposed scheme follows in most parts the standard approach of group signatures, integrates the pseudonym idea from DAA, and provides a new way to prove linkage of a *batch* of signatures.

3.1 Syntax

In group signatures, an *issuer* interacts with *users* who want to join the group and become group members. Members create anonymous signatures on behalf of the group, which verifiers can check without learning the signers' identity. In our setting, the anonymity of the signer is steered via *pseudonyms*, generated with every signature, as well as explicit *link proofs*. More precisely, a UCL scheme supports two types of linkability (see Fig. 1 for a pictorial representation):

Implicit Linkability: Every signature is accompanied with a pseudonym, generated by the user for a particular scope. Re-using the same scope leads to the same pseudonym, making all signatures generated for the same *scope* immediately linkable for the verifier. Pseudonymous signatures for different scopes cannot be linked, except via explicit link proofs generated by the user.

Explicit Linkability: After the signatures have been generated, they can be claimed and linked by the user: given a set of signatures, the user proves that she created all of them, i.e., links the signatures in the set.

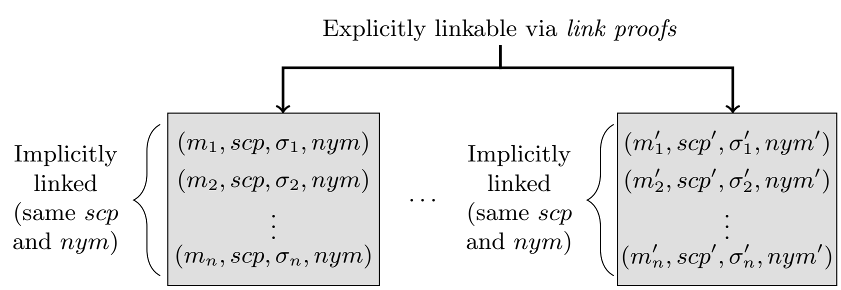


Fig. 1: Implicit vs explicit linkability on signatures by a same user, controlled by the user via scopes, pseudonyms and link proofs.

We emphasize that users have full control on the scopes, which can be any arbitrary (bit)string. For instance, in the contact tracing example given in Section

1, where identifiers are reused during 15 minutes, the scope could be derived from publicly available information, such as the current epoch. Alternatively, using randomly chosen scopes would lead to unlinkable signatures.

A UCL group signature scheme consists of the following algorithms:

$\mathsf{Setup}(1^\tau) \rightarrow param$**:** Generates the public parameters for the scheme.

$\mathsf{IKGen}(param) \rightarrow (isk, ipk)$**:** Generates the issuer's keypair (isk,ipk).

$\langle \mathsf{Join}(ipk), \mathsf{Issue}(ipk, isk) \rangle \rightarrow (usk, \perp)$**:** To become a member of the group, the user runs the interactive join protocol with the issuer. If successful, the user obtains a user secret key usk.

$\mathsf{Sign}(ipk, usk, m, scp) \rightarrow (\sigma, nym)$**:** Signs a message m w.r.t. scope scp via user secret key usk. The output is a pseudonym nym and group signature σ.

$\mathsf{Verify}(ipk, \Sigma) \rightarrow 0/1$**:** On input a group public key ipk and tuple $\Sigma = (m, scp, \sigma, nym)$, containing a group signature σ and a pseudonym nym, purportedly corresponding to m and scp, returns 1 when the tuple is valid and 0 otherwise.

$\mathsf{Link}(ipk, usk, lm, \boldsymbol{\Sigma}) \rightarrow \pi_l/\perp$**:** On input a set of signature tuples $\boldsymbol{\Sigma} = \{\Sigma_i\}_{i\in[n]}$ and user secret key usk, produces a proof π_l of these signatures being linked or $\perp$ indicating failure. The link proof is also done for a specific message lm, which can be used e.g., to ensure freshness of the proof.

$\mathsf{VerifyLink}(ipk, lm, \boldsymbol{\Sigma}, \pi_l) \rightarrow 0/1$**:** Returns 1 if π_l is a valid proof for the statement that $\boldsymbol{\Sigma} = \{\Sigma_i\}_{i\in[n]}$ were produced by the same signer and for link message lm, or 0 otherwise.

We delay the definition of the correctness properties for a UCL scheme after introducing some extra notation in the next section.

3.2 Security Model

A UCL group signature scheme should provide the following privacy and security properties: For privacy, signatures should not leak anything about the signer's identity beyond what is exposed by the user through implicit and explicit linkability (**anonymity**). Security is expressed through a number of properties covering the desired unforgeability guarantees: signatures should only be created by users that have correctly joined the group (**traceability**), and even a corrupt issuer should not be able to impersonate honest users (**non-frameability**).

Oracles and State. Our definitional framework closely follows the existing work of group signatures, and in particular the work by [5] for security of dynamic schemes. They make use of a number of oracles and global variables that allow the adversary to engage with honest parties, and which we adjust to our setting.

ADDU**:** Runs $\langle \mathsf{Join}, \mathsf{Issue} \rangle$ between an honest user and an honest issuer, allowing the adversary to enroll honest users. The new user key is stored as $\mathsf{USK}[\mathsf{uid}]$.

SNDU**:** (The SeND to User oracle.) Runs the Join process on behalf of an honest user, against an adversarially controlled issuer. The new user key is stored as $\mathsf{USK}[\mathsf{uid}]$.

SNDI: (The SeND to Issuer oracle.) Runs the Issue process on behalf of an honest issuer, allowing the adversary to join in the role of corrupt users in games with an honest issuer. Updates transcript[uid] with a transcript of the exchanged messages.

SIGN/LINK: Allow the adversary to obtain honest users' signatures/link proofs for messages/signatures of his choice (with restrictions in anonymity game).

CH-SIGN$_b$/CH-LINK$_b$: Challenge oracles in the anonymity game that allow the adversary to get signatures and link proofs for a challenge user uid$_b$.

Fig. 2 presents the details of the oracles used in our games: the standard ADDU, SNDU, and SNDI oracles as defined in [5], and SIGN and CH-SIGN$_b$, which we modify from [5], and LINK and CH-LINK$_b$, which are specific to our model.

Variable	Content
uid$_b^*$	Challenge user in anon-b. Ignored in the other games.
HUL	uids of honest users that have joined
CUL	uids of corrupt users that have joined (only needed when issuer is honest)
SIG[uid]	signature tuples (m, scp, σ, nym) produced by SIGN for user uid
CSIG	signatures tuples (m, scp, σ, nym) by uid$_b^*$ produced via CH-SIGN$_b$
LNK[uid]	link queries $(lm, \boldsymbol{\Sigma})$ sent to LINK for uid
CLNK	link queries $(lm, \boldsymbol{\Sigma})$ made to CH-LINK$_b$
USK[uid]	signing key of honest user uid
transcript[uid]	messages from join protocol between corrupt user uid & honest issuer

Table 1: Information stored by the global state variables.

Helper Function Identify. In some security games we need to determine if a certain user secret key was used to create a given signature. For this we follow DAA work [6,12] and assume the availability of a function Identify$(ipk, usk, \Sigma) \to 0/1$, returning 1 when $\Sigma = (m, scp, \sigma, nym)$ was produced by usk, or 0 otherwise.

We assume the following behaviour of Identify: for all $(isk, ipk) \leftarrow$ IKGen$(param)$, and all $\Sigma = (m, scp, \sigma, nym)$ where Verify$(ipk, \Sigma) = 1$ there must exist *exactly one* usk (from the user secret key space induced by $\langle$Join(ipk), Issue$(ipk, isk)\rangle$) such that Identify$(ipk, usk, \Sigma) = 1$.

We use the function for keys of both honest and corrupt users. Abusing notation, we write Identify(uid, Σ) to indicate that Identify is run for the secret key usk of user uid (where ipk is clear from the context). For honest users, Identify simply uses USK[uid]; while keys of corrupt users can be extracted from the join transcript. For the latter, note that Identify is only used in games where the issuer is honest, i.e., such a transcript is available. In our concrete scheme we exploit the random oracle to extract a user's keys via rewinding. If online-extractable proofs are used, then Identify will also receive the trapdoor information as input.

We now formally capture the expected security properties.

$\underline{\mathsf{ADDU}\ (\mathsf{uid})\ //\ \text{From [5]}}$

$\textbf{if } \mathsf{uid} \in \mathsf{HUL} \cup \mathsf{CUL} : \textbf{return } \bot$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{\mathsf{uid}\}$
$\mathsf{dec}^{\mathsf{uid}} \leftarrow \mathsf{cont}, st^{\mathsf{uid}}_{\mathsf{Join}} \leftarrow ipk, st^{\mathsf{uid}}_{\mathsf{Issue}} \leftarrow (isk, ipk)$
$(st^{\mathsf{uid}}_{\mathsf{Join}}, \mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^{\mathsf{uid}}) \leftarrow \mathsf{Join}(st^{\mathsf{uid}}_{\mathsf{Join}}, \bot)$
$\textbf{while } \mathsf{dec}^{\mathsf{uid}} = \mathsf{cont} :$
$\quad (st^{\mathsf{uid}}_{\mathsf{Issue}}, \mathsf{M}_{\mathsf{Join}}, \mathsf{dec}^{\mathsf{uid}}) \leftarrow \mathsf{Issue}(st^{\mathsf{uid}}_{\mathsf{Issue}}, \mathsf{M}_{\mathsf{Issue}})$
$\quad \textbf{if } \mathsf{dec}^{\mathsf{uid}} = \mathsf{accept} : \mathsf{transcript}[\mathsf{uid}] \leftarrow st^{\mathsf{uid}}_{\mathsf{Issue}}$
$\quad (st^{\mathsf{uid}}_{\mathsf{Join}}, \mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^{\mathsf{uid}}) \leftarrow \mathsf{Join}(st^{\mathsf{uid}}_{\mathsf{Join}}, \mathsf{M}_{\mathsf{Join}})$
$\textbf{if } \mathsf{dec}^{\mathsf{uid}} = \mathsf{accept} : \mathsf{USK}[\mathsf{uid}] \leftarrow st^{\mathsf{uid}}_{\mathsf{Join}}$
$\textbf{return } \mathsf{accept}$

$\underline{\mathsf{SIGN}(\mathsf{uid}, m, scp)\ //\ \text{Modified from [5]}}$

$\textbf{if } \mathsf{uid} \notin \mathsf{HUL} \vee \mathsf{USK}[\mathsf{uid}] = \bot : \textbf{return } \bot$
$(\sigma, nym) \leftarrow \mathsf{Sign}(ipk, \mathsf{USK}[\mathsf{uid}], m, scp)$
$\Sigma \leftarrow (m, scp, \sigma, nym), \mathsf{SIG}[\mathsf{uid}] \leftarrow \mathsf{SIG}[\mathsf{uid}] \cup \{\Sigma\}$
$\textbf{return } (\sigma, nym)$

$\underline{\mathsf{CH\text{-}SIGN}_b(m, scp)\ //\ \text{Modified from [5]}}$

// Initialized with $\mathsf{uid}^*_b, \mathsf{uid}^*_{1-b}$ by the experiment
$(\sigma, nym) \leftarrow \mathsf{Sign}(ipk, \mathsf{USK}[\mathsf{uid}^*_b], m, scp)$
$\Sigma \leftarrow (m, scp, \sigma, nym), \mathsf{CSIG} \leftarrow \mathsf{CSIG} \cup \{\Sigma\}$
$\textbf{return } (\sigma, nym)$

$\underline{\mathsf{CH\text{-}LINK}_b(lm, \boldsymbol{\Sigma})\ //\ \text{New w.r.t. [5]}}$

// Initialized with uid^*_b by the experiment
$\mathsf{CLNK} \leftarrow \mathsf{CLNK} \cup (lm, \boldsymbol{\Sigma})$
$\pi_l \leftarrow \mathsf{Link}(ipk, \mathsf{USK}[\mathsf{uid}^*_b], lm, \boldsymbol{\Sigma})$
$\textbf{return } \pi_l$

$\underline{\mathsf{SNDU}(\mathsf{uid}, \mathsf{M}_{in})\ //\ \text{From [5]}}$

$\textbf{if } \mathsf{uid} \notin \mathsf{HUL} :$
$\quad \mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{\mathsf{uid}\}$
$\quad \mathsf{M}_{in} \leftarrow \bot, \mathsf{dec}^{\mathsf{uid}} \leftarrow \mathsf{cont}$
$\textbf{if } \mathsf{dec}^{\mathsf{uid}} \neq \mathsf{cont} : \textbf{return } \bot$
$\textbf{if } st^{\mathsf{uid}}_{\mathsf{Join}} = \bot : st^{\mathsf{uid}}_{\mathsf{Join}} \leftarrow ipk$
$(st^{\mathsf{uid}}_{\mathsf{Join}}, \mathsf{M}_{out}, \mathsf{dec}^{\mathsf{uid}}) \leftarrow \mathsf{Join}(st^{\mathsf{uid}}_{\mathsf{Join}}, \mathsf{M}_{in})$
$\textbf{if } \mathsf{dec}^{\mathsf{uid}} = \mathsf{accept} : \mathsf{USK}[\mathsf{uid}] \leftarrow st^{\mathsf{uid}}_{\mathsf{Join}}$
$\textbf{return } (\mathsf{M}_{out}, \mathsf{dec}^{\mathsf{uid}})$

$\underline{\mathsf{SNDI}\ (\mathsf{uid}, \mathsf{M}_{in})\ //\ \text{From [5]}}$

$\textbf{if } \mathsf{uid} \in \mathsf{HUL} : \textbf{return } \bot$
$\textbf{if } \mathsf{uid} \notin \mathsf{CUL} :$
$\quad \mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{\mathsf{uid}\}, \mathsf{dec}^{\mathsf{uid}} \leftarrow \mathsf{cont}$
$\textbf{if } \mathsf{dec}^{\mathsf{uid}} \neq \mathsf{cont} : \textbf{return } \bot$
$\textbf{if } st^{\mathsf{uid}}_{\mathsf{Issue}} = \bot : st^{\mathsf{uid}}_{\mathsf{Issue}} \leftarrow (isk, ipk)$
$(st^{\mathsf{uid}}_{\mathsf{Issue}}, \mathsf{M}_{out}, \mathsf{dec}^{\mathsf{uid}}) \leftarrow \mathsf{Issue}(st^{\mathsf{uid}}_{\mathsf{Issue}}, \mathsf{M}_{in})$
$\textbf{if } \mathsf{dec}^{\mathsf{uid}} = \mathsf{accept} :$
$\quad \mathsf{transcript}[\mathsf{uid}] \leftarrow st^{\mathsf{uid}}_{\mathsf{Issue}}$
$\textbf{return } (\mathsf{M}_{out}, \mathsf{dec}^{\mathsf{uid}})$

$\underline{\mathsf{LINK}(\mathsf{uid}, lm, \boldsymbol{\Sigma})\ //\ \text{New w.r.t. [5]}}$

$\textbf{if } \mathsf{uid} \notin \mathsf{HUL} \vee \mathsf{USK}[\mathsf{uid}] = \bot : \textbf{return } \bot$
$\mathsf{LNK}[\mathsf{uid}] \leftarrow \mathsf{LNK}[\mathsf{uid}] \cup (lm, \boldsymbol{\Sigma})$
$\pi_l \leftarrow \mathsf{Link}(ipk, \mathsf{USK}[\mathsf{uid}], lm, \boldsymbol{\Sigma})$
$\textbf{return } \pi_l$

Fig. 2: Detailed oracles available in our model.

Correctness. We formalize the correctness of Sign and correctness of Link properties in Appendix B.1.

Anonymity. We adapt the classic privacy notion to our setting. It expresses that signatures must not reveal anything about the signer's identity beyond what was intended by her, even when the issuer is corrupt. The adversary plays the role of the issuer and can trigger honest users to join, sign and link. Eventually, he chooses two honest users uid_0^* and uid_1^*, and one becomes the challenge user uid_b^*. The adversary can receive signatures and link proofs of uid_b^* (via $\mathsf{CH\text{-}SIGN}_b$ and $\mathsf{CH\text{-}LINK}_b$) and must determine b better than by random guessing.

As our signatures support user-controlled linkability, we must be careful to exclude trivial wins leveraging it. There are two ways in which the adversary can trivially win. First, by leveraging implicit linkability: signatures by the same user and with the same scope are directly linkable. The adversary could exploit this by calling $\mathsf{CH\text{-}SIGN}_b$ and SIGN (the latter, for uid_0^* or uid_1^*) with the same scope. Second, the adversary can leverage explicit linkability by obtaining link proofs via LINK or $\mathsf{CH\text{-}LINK}_b$ for a set of signatures that contains challenge signatures, obtained though $\mathsf{CH\text{-}SIGN}_b$, and non-challenge signatures (for a challenge user), obtained from SIGN.

Definition 1. *(Anonymity). A group signature scheme* UCL *with user-controlled linkability is anonymous if for all ppt adversaries* $\mathcal{A}$*, the following is negligible in* τ*:* $|\Pr[\mathbf{Exp}^{\mathsf{anon\text{-}1}}_{\mathcal{A},\mathsf{UCL}}(\tau)=1]-\Pr[\mathbf{Exp}^{\mathsf{anon\text{-}0}}_{\mathcal{A},\mathsf{UCL}}(\tau)=1]|$.

Experiment: $\mathbf{Exp}^{\mathsf{anon}\text{-}b}_{\mathcal{A},\mathsf{UCL}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}_0^*, \mathsf{uid}_1^*, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{LINK}}(\text{choose}, ipk, isk)$
if $\mathsf{USK}[\mathsf{uid}_d^*] \neq \perp$ for $d = 0, 1$:
  Initialize $\mathsf{CH\text{-}SIGN}_b$ and $\mathsf{CH\text{-}LINK}_b$ with uid_b^*
else :
  return $\perp$
$b' \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{LINK},\mathsf{CH\text{-}SIGN}_b,\mathsf{CH\text{-}LINK}_b}(\text{guess}, \mathsf{state})$
// Trivial wins via implicit linking:
// $\mathcal{A}$ used the same scope in $\mathsf{CH\text{-}SIGN}_b$ and SIGN for one of the challenge user
if $\exists(*, scp, *) \in \mathsf{CSIG} \wedge \exists(*, scp, *) \in \mathsf{SIG}[\mathsf{uid}_d^*]$ for $d \in \{0, 1\}$:
  return $\perp$
// Trivial wins via explicit linking:
// $\mathcal{A}$ queried LINK or $\mathsf{CH\text{-}LINK}_b$ with both challenge and non-challenge sigs.
if $\exists \boldsymbol{\Sigma}$ s.t. $(\boldsymbol{\Sigma} \cap \mathsf{CSIG} \neq \emptyset \wedge (*, \boldsymbol{\Sigma}) \in \mathsf{LNK}[*]) \vee$
  $(\boldsymbol{\Sigma} \cap \mathsf{SIG}[\mathsf{uid}_d^*] \neq \emptyset \wedge (*, \boldsymbol{\Sigma}) \in \mathsf{CLNK}$ for $d \in \{0, 1\})$:
  return $\perp$
return b'

Traceability. This property covers the desired unforgeability guarantees for corrupt users of groups with an honest issuer. Intuitively, it guarantees that only legitimate members of the group are able to generate valid signatures on behalf of that group. The traditional approach in group signature models [5,25] is to ask the adversary for a forgery and leverage the trusted opener to check whether the forged signature opens to any user that has joined the group.

As our setting does not have such an opening entity, we cannot follow this approach and instead take inspiration from the DAA security models [6,12]. Therein, one uses the implicit availability of an Identify function (introduced above) which allows to check whether a given signature belongs to a certain user secret key (which we know from honest users, and can extract from corrupt ones). The adversary wins if he can produce valid signatures (or link proofs) that cannot be traced back via Identify to any member of the group. This alone would not be sufficient though, as our signatures also carry some information in their implicit and explicit linkability, which an adversary should not be able to manipulate either. That is, the adversary also wins if he can produce more standalone signatures that are unlinkable (for the same scope) than he controls corrupt users, or if he manages to produce a valid link proof for signatures of different corrupt users.

We have grouped these properties along the statement that the adversary has to forge, i.e., we have signature traceability for forgeries of standalone signatures, and link traceability that works analogously for the link proofs.

Definition 2. *(Signature Traceability). A group signature scheme* UCL *with user-controlled linkability provides signature traceability if for all ppt adversaries* $\mathcal{A}$, $|\Pr[\mathbf{Exp}^{\textsf{sign-trace}}_{\mathcal{A},\textsf{UCL}}(\tau)=1]|$ *is negligible in* τ.

Definition 3. *(Link Traceability). A group signature scheme* UCL *with user-controlled linkability provides link traceability if for all ppt adversaries* $\mathcal{A}$*, the following is negligible in* τ*:* $|\Pr[\mathbf{Exp}^{\textsf{link-trace}}_{\mathcal{A},\textsf{UCL}}(\tau)=1]|$.

Experiment: $\mathbf{Exp}^{\textsf{sign-trace}}_{\mathcal{A},\textsf{UCL}}(\tau)$

$param \leftarrow \textsf{Setup}(1^\tau), (ipk, isk) \leftarrow \textsf{IKGen}(param)$

$(\Sigma_1, \ldots, \Sigma_n) \leftarrow \mathcal{A}^{\textsf{ADDU,SNDI,SIGN,LINK}}(ipk)$

return 1 **if** :

$\forall i : \textsf{Verify}(ipk, \Sigma_i) = 1 \ \wedge \ \Sigma_i = (m_i, scp, \sigma_i, nym_i)$ // the scope is the same in all sigs

and one of the following conditions holds:

// Signature of non-member

1) $\exists \Sigma_i$ s.t. $\forall \textsf{uid} \in \textsf{HUL} \cup \textsf{CUL} : \textsf{Identify}(\textsf{uid}, \Sigma_i) = 0$

// More unlinkable sigs than corrupt users

2) $\forall i, j : nym_i \neq nym_j \ \wedge \ \Sigma_i \notin \textsf{SIG}[*] \ \wedge \ |\textsf{CUL}| < n$

Experiment: $\mathbf{Exp}_{\mathcal{A},\mathsf{UCL}}^{\text{link-trace}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$

$(lm, \boldsymbol{\Sigma}, \pi_l) \leftarrow \mathcal{A}^{\mathsf{ADDU,SNDI,SIGN,LINK}}(ipk)$

return 1 **if** :

$\mathsf{VerifyLink}(ipk, lm, \boldsymbol{\Sigma}, \pi_l) = 1$

and one of the two conditions holds:

// Contains signature of non-member

$1)\exists \Sigma \in \boldsymbol{\Sigma}$ s.t. $\forall \mathsf{uid} \in \mathsf{HUL} \cup \mathsf{CUL} : \mathsf{Identify}(\mathsf{uid}, \Sigma) = 0$

// sigs by different users

$2)\exists \mathsf{uid} \neq \mathsf{uid}', \Sigma \neq \Sigma' \in \boldsymbol{\Sigma}$ s.t. $\mathsf{Identify}(\mathsf{uid}, \Sigma) = 1 \wedge \mathsf{Identify}(\mathsf{uid}', \Sigma') = 1$

Non-Frameability. This property guarantees that an honest user cannot be framed by the adversary, even when the issuer is corrupt. In our setting such framing can be done when signatures of an honest user are linkable to signatures that she has not generated. As we support two different types of linkability, we again need a dedicated variant of that property for each of them. The first captures non-frameability from standalone signatures, i.e., via implicit linking. In this case, the adversary can only frame an honest user by producing a signature that holds for the same pseudonym that an honest signature generated for that scope. Linkability (and thus framing attacks) across scopes is not possible and thus does not have to be considered here. Such linkage for different scopes is only possible via explicit link proofs. The second property we define captures non-frameability for these proofs, which the adversary can leverage to frame an honest user in two ways: producing a proof that (1) links honestly generated signatures with adversarial ones; or (2) producing a proof that links honestly generated signatures by the same user, but the honest user did not create that proof – i.e., it is the proof itself that is forged and aims to impersonate the honest user.

Definition 4. *(Signature Non-frameability). A group signature scheme* UCL *with user-controlled linkability is secure against signature framing if for all ppt adversaries $\mathcal{A}$, the following is negligible in τ:* $|\Pr[\mathbf{Exp}_{\mathcal{A},\mathsf{UCL}}^{\text{sign-frame}}(\tau) = 1]|$.

Definition 5. *(Link Non-frameability). A group signature scheme* UCL *with user-controlled linkability is secure against link framing if for all ppt adversaries $\mathcal{A}$, the following is negligible in τ:* $|\Pr[\mathbf{Exp}_{\mathcal{A},\mathsf{UCL}}^{\text{link-frame}}(\tau) = 1]|$.

Experiment: $\mathbf{Exp}_{\mathcal{A},\mathsf{UCL}}^{\text{sign-frame}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$

$(\Sigma = (m, scp, \sigma, nym)) \leftarrow \mathcal{A}^{\mathsf{SNDU,SIGN,LINK}}(ipk, isk)$

return 1 **if** :

$\mathsf{Verify}(ipk, \Sigma) = 1$ and :

$\exists \mathsf{uid}$ s.t. $\Sigma \notin \mathsf{SIG}[\mathsf{uid}] \ \wedge \ (*, scp, *, nym) \in \mathsf{SIG}[\mathsf{uid}]$

Experiment: $\mathbf{Exp}_{\mathcal{A},\mathsf{UCL}}^{\text{link-frame}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(lm, \boldsymbol{\Sigma}, \pi_l) \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{LINK}}(ipk, isk)$
return 1 **if** :
  $\mathsf{VerifyLink}(ipk, lm, \boldsymbol{\Sigma}, \pi_l) = 1$
  and one of the following conditions hold:
    // Contains honest and adversarial sigs.
    1) $\exists \mathsf{uid}$ s.t. $\exists \Sigma, \Sigma' \in \boldsymbol{\Sigma} : \Sigma \in \mathsf{SIG}[\mathsf{uid}] \wedge \Sigma' \notin \mathsf{SIG}[\mathsf{uid}]$
    // Honestly created sigs., but π_l was forged
    2) $\exists \mathsf{uid}$ s.t. $\forall \Sigma \in \boldsymbol{\Sigma}, \Sigma \in \mathsf{SIG}[\mathsf{uid}] \ \wedge (lm, \boldsymbol{\Sigma}) \notin \mathsf{LNK}[\mathsf{uid}]$

Definition 6. *(Security of* UCL*). A group signature scheme* UCL *with user-controlled linkability is secure if it ensures the previous anonymity, traceability and non-frameability properties.*

3.3 Construction

We now present our scheme satisfying the desired security and privacy properties. The core of our constructions follows the standard approach of group signatures (see, e.g., [8]): during join, users receive from the issuer a membership credential, and signing essentially is a proof of knowledge of such a credential. We use BBS+ signatures for such blindly issued membership credentials.

Adding **implicit linkability:** Whereas standard group signatures usually include an encryption of the user's identity (for opening) in her signature, we use the pseudonym idea of DAA and anonymous credentials instead [6,12,14] and, specifically, of [11]. That is, when creating a signature, the user also reveals a pseudonym $nym \leftarrow \mathsf{H}(scp)^y$ for her key y and a particular scope scp. Clearly, these pseudonyms are scope-exclusive, i.e., there is only one valid pseudonym per scope and user key [14]. The user also proves that she has computed the pseudonym from her key.

Adding **explicit linkability:** The existing solution for link proofs [25,14] of signatures with different pseudonyms is to let the user provide a fresh proof that all pseudonyms are all based on the same user key. So far, this approach has been proposed for linking only two signatures, and will grow linearly when being used for many signatures. For our proofs, we instead use the observation that all individual pseudonyms the signatures are associated to can form a "meta-nym" $\overline{nym} = \prod_{i\in[n]} nym_i = \prod_{i\in[n]} \mathsf{H}(scp_i)^y$. That is, the user can simply prove that she knows the secret key y such that $\overline{nym} \leftarrow \overline{hscp}^y$, where $\overline{nym}$ and $\overline{hscp} = \prod_{i\in[n]} \mathsf{H}(scp_i)$ are uniquely determined by the signatures.

We stress that we do not claim novelty of the main parts of the group signatures. The core contribution here is (1) the simple trick for making efficient batched link proofs, and (2) making the pseudonym idea of credentials and DAA also formally available for group signatures.

Our Construction Π_{UCL}. Our concrete construction works as follows:

$\mathsf{Setup}(1^\tau) \to param$. Generates a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathsf{PG}(1^\tau)$ and two further generators $h_1, h_2 \in \mathbb{G}_1$ (for the BBS+ credentials).

$\mathsf{IKGen}(param) \to (isk, ipk)$. Outputs $isk \leftarrow_\$ \mathbb{Z}_p^*$ and $ipk \leftarrow g_2^{isk}$.

$\langle \mathsf{Join}(ipk), \mathsf{Issue}(ipk, isk) \rangle \to (usk, \perp)$. This interactive protocol lets the user blindly obtain a BBS+ signature by the issuer on her secret key y:

- Issuer: sends a random nonce $n \leftarrow \mathbb{Z}_p^*$ to the user.
- User: $y \leftarrow_\$ \mathbb{Z}_p^*, Y \leftarrow h_1^y$, $\pi_Y \leftarrow \mathsf{SPK}\{(y) : Y \leftarrow h_1^y\}((param, h_1, Y), n)$. Sends (Y, π_Y) back to the issuer.
- Issuer: Only proceeds if π_Y is valid. Computes BBS+ signature on y as $x, s \leftarrow_\$ \mathbb{Z}_p^*$, $A \leftarrow (Y h_2^s g_1)^{1/(isk+x)}$. Sends (A, x, s) to user.
- User: If $A \neq 1_{\mathbb{G}_1}, e(A, g_2)^x e(A, ipk) = e(g_1 Y h_2^s, g_2)$ outputs $usk \leftarrow (A, x, y, s)$.

$\mathsf{Sign}(ipk, usk, m, scp) \to (\sigma, nym)$. To sign a message m for scope scp, the user generates the pseudonym $nym \leftarrow \mathsf{H}(scp)^y$ and computes a proof that the pseudonym was computed for a key that she has a BBS+ credential on, including the message m in the Fiat-Shamir hash of the proof.

- Parse usk as (A, x, y, s).
- Compute the pseudonym as: $nym \leftarrow \mathsf{H}(scp)^y$.
- Re-randomize the BBS+ credential as $r_1, r_2 \leftarrow_\$ \mathbb{Z}_p^*$, $r_3 \leftarrow r_1^{-1}$ and $s' \leftarrow s - r_2 r_3$, $A' \leftarrow A^{r_1}$, $\hat{A} \leftarrow (A')^{-x}(g_1 h_1^y h_2^s)^{r_1}$, $d \leftarrow (g_1 h_1^y h_2^s)^{r_1} h_2^{-r_2}$.
- Compute $\pi_\sigma \leftarrow \mathsf{SPK}\{(x, y, r_2, r_3, s') : nym = \mathsf{H}(scp)^y \wedge$
  $$\hat{A}/d = (A')^{-x} h_2^{r_2} g_1 h_1^y = d^{r_3} h_2^{-s'}\}(ctx, m)$$
  for $ctx \leftarrow (param, A', \hat{A}, d, nym)$.
- $\sigma \leftarrow (A', \hat{A}, d, \pi_\sigma)$. Return (σ, nym).

$\mathsf{Verify}(ipk, \Sigma)$. Parses σ in Σ as $(A', \hat{A}, d, \pi_\sigma)$, checks that $A' \neq 1_{\mathbb{G}_1}$, $e(A', ipk) = e(\hat{A}, g_2)$,and outputs 1 if the SPK in Σ is valid for message m and scope scp.

$\mathsf{Link}(ipk, lm, \boldsymbol{\Sigma}) \to \pi_l/\perp$. Linking signatures is done by batching all nyms and scopes into $\overline{nym}$ and $\overline{hscp}$, and proving knowledge of the discrete logarithm of $\overline{nym}$ w.r.t. $\overline{hscp}$. The link message lm is included in the hash of the proof.

- Parse usk as (A, x, y, s), and $\boldsymbol{\Sigma}$ as $\{\Sigma_i = (m_i, scp_i, \sigma_i, nym_i)\}_{i \in [n]}$.
- If $\exists i \in [n]$ s.t. $\mathsf{H}(scp_i)^y \neq nym_i$, or $\mathsf{Verify}(ipk, \Sigma_i) = 0$, return $\perp$.
- Set $ctx \leftarrow (param, \{scp_i\}_{i \in [n]}, \{nym_i\}_{i \in [n]})$.
- Compute $\overline{hscp} \leftarrow \prod_{i \in [n]} \mathsf{H}(scp_i)$ and $\overline{nym} \leftarrow \overline{hscp}^y$.
- Output $\pi_l \leftarrow \mathsf{SPK}\{(y) : \overline{nym} = \overline{hscp}^y\}(ctx, lm)$.

$\mathsf{VerifyLink}(ipk, lm, \boldsymbol{\Sigma}, \pi_l) \rightarrow 0/1$. The verifier recomputes the meta-scope $\overline{hscp}$ and meta-nym $\overline{nym}$ from the individual signatures, verifies all signatures and π_l:

- Parse $\boldsymbol{\Sigma}$ as $\{\Sigma_i = (m_i, scp_i, \sigma_i, nym_i)\}_{i \in [n]}$.
- If $\exists i \in [n]$ s.t. $\mathsf{Verify}(ipk, \Sigma_i) = 0$, return 0.
- If $\exists i \neq j \in [n]$ s.t. $scp_i = scp_j \wedge nym_i \neq nym_j$, return 0.
- $\overline{hscp} = \prod_{i \in [n]} \mathsf{H}(scp_i)$, $\overline{nym} = \prod_{i \in [n]} nym_i$.
- Output result of verifying π_l for $\overline{hscp}$ and $\overline{nym}$.

3.3.1 Security of our Construction

Theorem 1. *Assuming* SPK *is zero-knowledge and simulation-sound, our construction is secure under the discrete logarithm, DDH, and q-SDH assumptions, in the random oracle model for* H *and* SPK*.*

Proof sketch. Under the DDH assumption [27], *anonymity* follows from zero-knowledgeness and simulation-soundness of the SPKs, and the fact that pseudonyms are indistinguishable from random when different scopes are used.

We realize Identify with the help of the pseudonyms. Given a signature (m, scp, σ, nym), Identify fetches y from the *usk* of the specified uid and, if $\mathsf{H}(scp)^y = nym$, returns 1; else, returns 0. Scope-exclusiveness of pseudonyms ensures the required uniqueness [14]. Then, *signature traceability* follows from unforgeability of the BBS+ credentials, and zero-knowledgeness and soundness of SPK: if the adversary produces, for the same scope, more unlinkable signatures than corrupt users, or a signature from a non-member, we extract a forged BBS+ credential and can break the q-SDH assumption [11]. Winning condition 1 of *link traceability* is shown similarly. For condition 2, soundness of SPK ensures the individual signatures and the link proof are valid discrete logarithm proofs. Also, after the uniqueness property of pseudonyms, no two nyms in the same link proof can have different values if derived from the same *scp*. This prevents malleability attacks: e.g., corrupt users joining with $y = a$ and $y = b - a$ and using nyms derived from those keys and the same *scp* in the same link proof. Thus, an adversary can only try to subvert the proof with nyms derived from different scopes. But this requires to find non-trivial roots in an equation of the form $g^{\alpha_1 y_1} ... g^{\alpha_n y_n} = 1$, where the y_i's are controlled by the adversary, but the α_i's are not, as the g^{α_i}'s are produced by H (a random oracle). We show that a successful adversary can be used to break the discrete logarithm assumption.

For *signature non-frameability*, we rely on the uniqueness property of the pseudonyms and zero-knowledgeness and soundness of SPK. We break the discrete logarithm assumption from an adversary forging a signature with the same scope and nym that a signature of an honest user. For *link non-frameability*, we rely on the zero-knowledgeness and soundness of SPK. First, a similar argument as in traceability ensures that the link proof must be over the same exponents. We leverage this to embed a DL challenge into the nyms and link proofs of an honest user. If the adversary forges a signature (for winning condition 1) or a link proof (winning condition 2) for this user, we can extract a solution to the challenge.

The full proofs are given in Appendix B.

3.3.2 Leveraging a Trusted Bulletin Board. Our UCL group signatures target a setting where signatures are generated and collected in a pseudonymous manner, and where linkability can still be refined later on by the users. Such a setting implicitly assumes the storage and availability of the originally exposed group signatures, e.g., in form of a central data lake that collects all individual signatures. In applications where the data lake is trusted by the verifiers (or even maintained by them), we can leverage this to improve the efficiency of our scheme. For clarity, we refer to such a trusted data lake and the additional functionality it must provide as *bulletin board* (BB), which can be used as follows:

- All signatures Σ_i are sent to the BB, who verifies and appends them, if valid.
- Link and VerifyLink no longer check the validity of all Σ_i in $\boldsymbol{\Sigma}$, but simply check whether all signatures are in the BB.

By using such a trusted BB we can improve the efficiency of Link and VerifyLink significantly – of course for the price of trusting a central entity again. This trust assumption would be necessary for the anonymity, link traceability and link non-frameability properties. However, the functionality of the BB can easily be distributed, e.g., using a blockchain; or the trust enforced and verified via regular audits where verifiers randomly pick signatures in the BB and check their validity. Thus, we believe that such a trust assumption is much more relaxed than trusting an entity that can single-handedly revoke the anonymity of all users.

Requirements on long-term storage capacity of the bulletin board depend on the use case. However, it seems reasonable to assume that, for most real world settings, a maximum timespan for storing past signatures can be established.

4 Scheme with Sequential Linkability (sUCL)

We extend our basic UCL scheme to allow for sequential link proofs. These sequential proofs target a setting where the originally signed (and unlinkable) data has an inherent order, e.g., time series data when sensors or vehicles continuously upload their measurements into a data lake. While the data is collected in unlinkable form, the eventual subsequent link proof must re-establish not only the correlation but also the order of a selected subset in an immutable manner.

We start by describing the minor syntax changes needed for our sequential group signatures (sUCL), and then discuss the additional security property we want such a sUCL scheme to achieve. Roughly, when making a sequential link proof, a corrupt user should not be able to swap, omit or insert signatures within the selected interval – and yet, this proves, nor reveals, nothing about signatures *outside* the proven interval. For this *sequentiality* property, we consider security against honest-then-corrupt users. While this may seem too lenient, note that it fits many real world applications where signing is an automatic process performed in the background by some device or application. In those cases, the need to alter sequences will only arise *after* the signatures have been created and sent. But, as described, the produced signatures – which contain extra information to enable proving order – are assumed to be stored in a data lake. Then, eventually, users have to make some claim that involves proving order with respect to those previously stored signatures. But this limits the options of malicious users. E.g., assume signatures Σ_1, Σ_2 and Σ_3 are produced in that order (i.e., first Σ_1, then Σ_2 and finally Σ_3), but a malicious user $\mathcal{A}$ wants to prove the reverse order. Then, $\mathcal{A}$ needs to commit to that strategy *before* sending the signatures by consequently altering the order information embedded in the signatures. Our argument is that, in many real world cases, $\mathcal{A}$ will not know which order he will be interested to prove in the future. For instance, in a contact tracing scenario (for a pandemic), malicious users will not know what order they are interested to prove until *after* learning which has been the risky contact.

Moreover, which specific alteration might be needed would also depend on the originally produced (and signed) data, and uninformed/random alterations may very well be useless or even counterproductive for the purposes of a malicious user. Nevertheless, even modeling this weak property requires a non-trivial approach. In Section 6, we give some insight about what seems to be possible beyond the honest-then-corrupt approach.

Finally, we present a simple extension to our Π_{UCL} scheme that uses the trusted bulletin board sketched in Section 3.3.2 and includes a hidden hash-chain into the group signatures, which allows to re-establish the order of signatures.

Syntax of sUCL. The signatures — despite being unlinkable per se — must now have an implicit order that can be recovered and verified through SLink and VerifySLink respectively. Abusing notation, we consider the set of signatures $\boldsymbol{\Sigma}_o$ to be given as an *ordered* set, and the proof and verification is done with respect to. this order. Further, to allow signatures to have an implicit order, we need to turn SSign into a stateful algorithm. That is, in addition to the standard input, it also receives a state st and outputs an updated state st'. We model that the state is initially set together with usk during the Join protocol. In summary, a sUCL scheme follows the UCL syntax from Section 3.1 with the following modifications:

$\langle \mathsf{Join}(ipk), \mathsf{Issue}(ipk, isk)\rangle \rightarrow ((usk, st), \perp)$**:** Initializes user state st.
$\mathsf{SSign}(ipk, usk, st, m, scp) \rightarrow ((\tilde{\sigma}, nym), st')$**:** Stateful sign algorithm.
$\mathsf{SLink}(ipk, usk, lm, \boldsymbol{\Sigma}_o) \rightarrow \pi_{seq}/\perp$**:** Sequential link proof for the ordered set $\boldsymbol{\Sigma}_o$.
$\mathsf{VerifySLink}(ipk, lm, \boldsymbol{\Sigma}_o, \pi_{seq}) \rightarrow 0/1$**:** Verifies π_{seq} w.r.t. the order in $\boldsymbol{\Sigma}_o$.

4.1 Security Model for sUCL

We want the sUCL scheme to have (essentially) the same traceability, non-frameability and anonymity properties as in Section 3.2 — and additionally guarantee the correctness and security of the re-established sequential order.

Traceability and Non-frameability. These properties cover the security expected through the controlled linkage (not order) and only need minor adjustments to cater for the changed syntax: In the games, we use $\mathsf{SSIGN}/\mathsf{SLINK}$ instead of $\mathsf{SIGN}/\mathsf{LINK}$.

4.1.1 Sequentiality. This property captures the security we can expect from proofs that reveal the sequential order of several signatures issued by a same user. Namely, when a user makes a sequential link proof for an ordered set $\boldsymbol{\Sigma}_o = \Sigma_1, \ldots, \Sigma_n$, we want to ensure that $\Sigma_1, \ldots, \Sigma_n$ have occurred indeed in that order and that no signature is omitted or inserted. The latter prevents attacks where a corrupt user tries to "hide" or add certain signatures, e.g., when a driver is asked to reveal the speed measurements from a certain time interval and wants to omit the moment she was speeding.

We follow the classic unforgeability style of definition and ask the adversary to output a forged link proof with an incorrect sequence. Clearly, such a definition needs to be able to capture what the "right order" of signatures is, in order to quantify whether a forgery violates that order or not. To do so, we opted for a two-stage game where the adversary can engage with honest users and make them sign (and link) messages of his choice. This ensures that we know the correct order in which the signatures are generated. Eventually, the adversary picks one of the honest users uid^*, upon which uid^* becomes corrupted and the adversary receives her secret key and current state. The adversary wins if he outputs a valid sequential link proof that violates the sequence produced by the originally honest user, e.g., re-orders, omits or inserts signatures.

Clearly we must allow the adversary to possibly include maliciously generated signatures in his forgery, but must be careful to avoid trivial wins: as soon as we give the adversary the secret key of uid^* he can trivially (re-)generate signatures on behalf of the honest user. Thus, we ask the adversary to commit to a set of maliciously generated signatures $\boldsymbol{\Sigma}'$ *before* corrupting uid^* and request that his link forgery for alleged ordered signatures $\boldsymbol{\Sigma}^*$ must be a subset of $\boldsymbol{\Sigma}' \cup \mathsf{SIG}[\mathsf{uid}^*]$.

Definition 7. *(Sequentiality). A group signature scheme* sUCL *with user-controlled sequential linkability ensures sequentiality if for all ppt adversaries* $\mathcal{A}$*, the following is negligible in* τ*:* $|\Pr[\mathbf{Exp}^{\mathsf{sequential}}_{\mathcal{A},\mathsf{sUCL}}(\tau) = 1]|$.

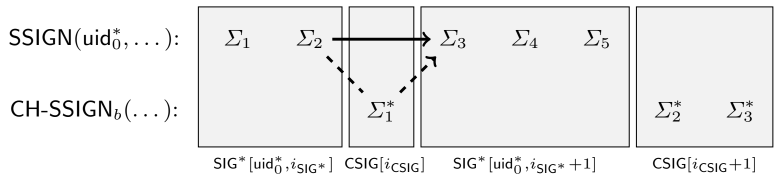
Experiment: $\mathbf{Exp}^{\mathsf{sequential}}_{\mathcal{A},\mathsf{sUCL}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}^*, \boldsymbol{\Sigma}', \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SSIGN},\mathsf{SLINK}}(\text{choose}, ipk)$
$\textbf{if } \mathsf{USK}[\mathsf{uid}^*] = \bot : \textbf{return } 0$
$\textbf{else } : \mathsf{HUL} \leftarrow \mathsf{HUL} \setminus \{\mathsf{uid}^*\}, \mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{\mathsf{uid}^*\}$
$/\!/$ $\mathsf{USK}[\mathsf{uid}^*]$ contains (usk, st) of uid^*
$(lm^*, \boldsymbol{\Sigma}^*, \pi^*_{seq}) \leftarrow \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SSIGN},\mathsf{SLINK}}(\text{forge}, \mathsf{state}, \mathsf{USK}[\mathsf{uid}^*])$
$\textbf{return } 1 \textbf{ if } :$
$\quad \mathsf{VerifySLink}(ipk, lm^*, \boldsymbol{\Sigma}^*, \pi^*_{seq}) = 1 \ \wedge$
$\quad \boldsymbol{\Sigma}^* \cap \mathsf{SIG}[\mathsf{uid}^*] \neq \emptyset \ \wedge$
$\quad \boldsymbol{\Sigma}^* \subseteq \boldsymbol{\Sigma}' \cup \mathsf{SIG}[\mathsf{uid}^*] \ \wedge$
$\quad \boldsymbol{\Sigma}^* \notin_o \mathsf{SIG}[\mathsf{uid}^*]$ $/\!/$ $\in_o$ means ordered check

4.1.2 Anonymity. In the basic scheme (UCL), we defined anonymity with the typical approach: the adversary first picks two honest users and must then guess which one is used to produce challenge signatures and link proofs. In UCL, we just needed to prevent the adversary from leveraging implicit linkability and explicit linkability. This boils down to not allowing the reuse of scopes between calls to $\mathsf{CH\text{-}SIGN}_b$ and SIGN (for challenge users), and not allowing to link signatures produced by $\mathsf{CH\text{-}SIGN}_b$ and SIGN (again, for challenge users).

In the sequential extension (sUCL), the idea is still the same, i.e., the adversary has to guess which is the chosen challenge user out of the two he picked up. However, the adversary has more ways to trivially learn the challenge user by leveraging the order information unavoidably revealed by the sequential link queries. Take, for instance, the scenario sketched in Fig. 3. There, the adversary interleaves a call to $\mathsf{CH\text{-}SSIGN}_b$ (the one producing Σ_1^*) between calls to SSIGN for the same challenge user (the call that produces Σ_2 and the calls producing Σ_3–Σ_5). If the adversary makes a call to SLINK with the signatures produced before and after the call to $\mathsf{CH\text{-}SSIGN}_b$ (e.g., including Σ_2, Σ_3 in Fig. 3) and the call fails, then the challenge user is the same as the one used in the calls to SSIGN. Indeed, the link call fails because one signature is missing in the sequence (and, in Fig. 3 the correct sequence would be the dashed one). Similarly, if the call succeeds, then the challenge user is not the one used in the calls to SSIGN (and the correct sequence in Fig. 3 is the solid one). Note that this works even when the scopes in all signatures are different: hence, it would not constitute a disallowed action in the UCL model. A similar strategy interleaving a call to SSIGN between calls to $\mathsf{CH\text{-}SSIGN}_b$ also applies.

Oracles and state. In the previous example, we saw that calls to $\mathsf{CH\text{-}SSIGN}_b$ and SSIGN (the latter for uid_0^* or uid_1^*) can later be used to (trivially) expose the challenge user – by linking signatures produced before those calls, with signatures produced after. However, linking signatures produced within the same interval of such calls should not leak any information about the challenge user. To capture



$$\Sigma_i = (m_i, scp_i, \sigma_i, nym_i)$$
$$\Sigma_i^* = (m_i^*, scp_i^*, \sigma_i^*, nym_i^*)$$

Fig. 3: Sketch of a strategy leading to a trivial win by $\mathcal{A}$ leveraging order information in sUCL, and the model to detect it.

those intervals, we assign every honestly generated signature to a cluster (set of signatures). Since the calls to $\mathsf{CH\text{-}SSIGN}_b$ and SSIGN are the events defining the linkage of which signatures would lead to trivial wins, we use those calls to mark when we need to start assigning signatures to a new cluster.

More specifically, to keep track of the cluster to which we need to assign signatures by challenge users, we resort to two counters: i_{SIG^*} and i_{CSIG}. Every time the adversary makes a call to $\mathsf{CH\text{-}SSIGN}_b$, we dump all signatures produced by $\mathsf{SSIGN}(\mathsf{uid}_b^*, \dots)$ since the last call to $\mathsf{CH\text{-}SSIGN}_b$ to a new cluster $\mathsf{SIG}^*[\mathsf{uid}_b^*, i_{\mathsf{SIG}^*}]$, and increment i_{SIG^*}. Similarly, when a call to $\mathsf{SSIGN}(\mathsf{uid}_b^*, \dots)$ is made, we increment i_{CSIG} so that all signatures produced by $\mathsf{CH\text{-}SSIGN}_b$ from that point onwards start being assigned to a new cluster $\mathsf{CSIG}[i_{\mathsf{CSIG}}]$.

In the example in Fig. 3, this restricts the adversary to making SLINK queries containing signatures in either $\mathsf{SIG}^*[\mathsf{uid}_0^*, i_{\mathsf{SIG}^*}]$, $\mathsf{CSIG}[i_{\mathsf{CSIG}}]$, $\mathsf{SIG}^*[\mathsf{uid}_0^*, i_{\mathsf{SIG}^*}+1]$, or $\mathsf{CSIG}[i_{\mathsf{CSIG}}+1]$, but not of any combination of (subsets of) those clusters.

The oracles used to model sUCL are summarized next and fully defined in Fig. 4. The state variables are summarized in Table 2. We emphasize that the new modifications only affect the anonymity property, while the other properties just need to adjust for the updated syntax.

- SSIGN/SLINK extend SIGN/LINK. SSIGN uses st_{uid}, the state of user uid, to call SSign, and updates it with the returned st'_{uid}. SLINK gets an ordered set.
- $\mathsf{CH\text{-}SSIGN}_b$/$\mathsf{CH\text{-}SLINK}_b$. Challenge oracles for the anonymity game, allowing the adversary to get signatures and link proofs for the challenge user.

Helper Function Adjacent. We rely on a helper function, $\mathsf{Adjacent}(\mathsf{LNK}[\mathsf{uid}], \mathsf{CLNK}) \to 0/1$. It explores LNK to check link queries for honest signatures and CLNK to check link queries for challenge signatures. It returns 1 if SLINK and $\mathsf{CH\text{-}SLINK}_b$ have been respectively queried with two sets of signatures that were sequentially generated, or 0 otherwise. This is an artifact of our specific construction rather than a general requirement, though. In Π_{sUCL}, given two adjacent signatures Σ_n, Σ_{n+1}, if Σ_n is included in a link proof and Σ_{n+1} in *another* link proof, it is possible to determine that they were sequentially issued. Consequently, if one is

Variable	Content
$\mathsf{SIG}[\mathsf{uid}]$	signature tuples $(m, scp, \tilde{\sigma}, nym)$ produced by SSIGN for user uid.
$\mathsf{SIG}^*[\mathsf{uid}^*_b, i]$	i-th cluster of signature tuples for uid^*_b produced by SSIGN.
$\mathsf{CSIG}[i]$	i-th cluster of challenge signature tuples $(m, scp, \tilde{\sigma}, nym)$.
i_{SIG^*}	Counter for SIG^* clusters. Incremented when $\mathsf{CH\text{-}SSIGN}_b$ is called.
i_{CSIG}	Counter for CSIG clusters. Incremented when SSIGN is called.

Table 2: New/modified global state variables in the sequential UCL scheme.

$\mathsf{SSIGN}(\mathsf{uid}, m, scp)$

if $\mathsf{uid} \notin \mathsf{HUL} \vee \mathsf{USK}[\mathsf{uid}] = \perp$: **return** $\perp$
$((\tilde{\sigma}, nym), st'_{\mathsf{uid}}) \leftarrow \mathsf{SSign}(ipk, \mathsf{USK}[\mathsf{uid}], st_{\mathsf{uid}}, m, scp)$
$\Sigma \leftarrow (m, scp, \tilde{\sigma}, nym)$
$\mathsf{SIG}[\mathsf{uid}] \leftarrow \mathsf{SIG}[\mathsf{uid}] \cup \{\Sigma\}$, $st_{\mathsf{uid}} \leftarrow st'_{\mathsf{uid}}$
// If anon game and challenge user,
// counter for challenge cluster gets incremented
if $\mathsf{uid} = \mathsf{uid}^*_d$ for $d \in \{0,1\}$: $i_{\mathsf{CSIG}} \leftarrow i_{\mathsf{CSIG}} + 1$
return $(\tilde{\sigma}, nym)$

$\mathsf{CH\text{-}SSIGN}_b(m, scp)$

// Initialized with uid^*_b by the experiment
$((\tilde{\sigma}, nym), st'_{\mathsf{uid}^*_b}) \leftarrow \mathsf{SSign}(ipk, \mathsf{USK}[\mathsf{uid}^*_b], st_{\mathsf{uid}^*_b}, m, scp)$
$\Sigma \leftarrow (m, scp, \tilde{\sigma}, nym)$
$\mathsf{CSIG}[i_{\mathsf{CSIG}}] \leftarrow \mathsf{CSIG}[i_{\mathsf{CSIG}}] \cup \{\Sigma\}$, $st_{\mathsf{uid}^*_b} \leftarrow st'_{\mathsf{uid}^*_b}$
// Create new sigs. cluster for challenge users
for $d = 0, 1$:
  $\mathsf{SIG}^*[\mathsf{uid}^*_d, i_{\mathsf{SIG}^*}] \leftarrow \mathsf{SIG}[\mathsf{uid}^*_d]$, $\mathsf{SIG}[\mathsf{uid}^*_d] \leftarrow \emptyset$
$i_{\mathsf{SIG}^*} \leftarrow i_{\mathsf{SIG}^*} + 1$
return $(\tilde{\sigma}, nym)$

$\mathsf{SLINK}(\mathsf{uid}, lm, \boldsymbol{\Sigma}_o)$

if $\mathsf{uid} \notin \mathsf{HUL} \vee \mathsf{USK}[\mathsf{uid}] = \perp$: **return** $\perp$
$\mathsf{LNK}[\mathsf{uid}] \leftarrow \mathsf{LNK}[\mathsf{uid}] \cup (lm, \boldsymbol{\Sigma}_o)$
$\pi_{seq} \leftarrow \mathsf{SLink}(ipk, \mathsf{USK}[\mathsf{uid}], lm, \boldsymbol{\Sigma}_o)$
return π_{seq}

$\mathsf{CH\text{-}SLINK}_b(lm, \boldsymbol{\Sigma}_o)$

// Initialized with uid^*_b by the experiment
$\mathsf{CLNK} \leftarrow \mathsf{CLNK} \cup (lm, \boldsymbol{\Sigma}_o)$
$\pi_{seq} \leftarrow \mathsf{SLink}(ipk, \mathsf{USK}[\mathsf{uid}^*_b], lm, \boldsymbol{\Sigma}_o)$
return π_{seq}

Fig. 4: Modified versions of the SIGN, SLINK, $\mathsf{CH\text{-}SIGN}_b$ and $\mathsf{CH\text{-}LINK}_b$ oracles.

a challenge signature and the other is not, it would be possible to trivially guess the bit b in the anonymity game. The Adjacent function is defined in Fig. 5.

$\mathsf{Adjacent}(\mathsf{LNK}[\mathsf{uid}], \mathsf{CLNK})$

if $\mathsf{uid} \notin \{\mathsf{uid}^*_0, \mathsf{uid}^*_1\}$: **return** 0
return 1 **if** $\exists (lm, \boldsymbol{\Sigma} = \{\Sigma_i\}_{i \in [n]}) \in \mathsf{LNK}[\mathsf{uid}], (lm', \boldsymbol{\Sigma}' = \{\Sigma'_i\}_{i \in [n']}) \in \mathsf{CLNK}$
  and one of the following conditions holds:
  1) Σ_0 was produced by SSIGN immediately after $\Sigma'_{n'}$ being produced by $\mathsf{CH\text{-}SSIGN}_b$
  2) Σ'_0 was produced by $\mathsf{CH\text{-}SSIGN}_b$ inmmediately after Σ_n being produced by SSIGN

Fig. 5: Definition of the helper function Adjacent.

Anonymity definition. Beyond the cumbersome changes required to prevent the new trivial wins, and the extra $\mathsf{Adjacent}$ check required by our specific construction, we capture anonymity in sUCL as in UCL. Specifically, the adversary controls the issuer and allows users to join, sign and link signatures. He chooses a pair of honest users, one of which is randomly picked to initialize the challenge oracles. Eventually, the adverary needs to guess which one of the users was chosen, task for which he can query again the oracles, subject to the restrictions described above. The formal definition is given next.

Definition 8. *(Anonymity). A group signature scheme* sUCL *with user-controlled sequential linkability ensures anonymity if for all ppt adversaries* $\mathcal{A}$*, the following is negligible in* τ*:* $|\Pr[\mathbf{Exp}^{\mathsf{sanon}\text{-}1}_{\mathcal{A},\mathsf{sUCL}}(\tau)=1]-\Pr[\mathbf{Exp}^{\mathsf{sanon}\text{-}0}_{\mathcal{A},\mathsf{sUCL}}(\tau)=1]|$.

Experiment: $\mathbf{Exp}^{\mathsf{sanon}\text{-}b}_{\mathcal{A},\mathsf{sUCL}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$

$(\mathsf{uid}_0^*, \mathsf{uid}_1^*, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SSIGN},\mathsf{SLINK}}(\text{choose}, ipk, isk)$

if $\mathsf{USK}[\mathsf{uid}_d^*] \neq \perp$ for $d = 0, 1$: Initialize $\mathsf{CH\text{-}SSIGN}_b$ and $\mathsf{CH\text{-}SLINK}_b$ with uid_b^*

else : **return** $\perp$

$b' \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SSIGN},\mathsf{SLINK},\mathsf{CH\text{-}SSIGN}_b,\mathsf{CH\text{-}SLINK}_b}(\text{guess}, \mathsf{state})$

if $\mathsf{Adjacent}(\mathsf{LNK}[\mathsf{uid}_d^*], \mathsf{CLNK}) = 1$ for $d \in \{0, 1\}$: **return** $\perp$

// Trivial wins via implicit linking: $\mathcal{A}$ used same scp in calls to SIGN and $\mathsf{CH\text{-}SSIGN}_b$

if $\exists(*, scp, *) \in \bigcup_{\forall i_{\mathsf{CSIG}}} \mathsf{CSIG}[i_{\mathsf{CSIG}}] \wedge \quad \exists(*, scp, *) \in \mathsf{SIG}[\mathsf{uid}_d^*] \bigcup_{\forall i_{\mathsf{SIG}^*}} \mathsf{SIG}^*[\mathsf{uid}_d^*, i_{\mathsf{SIG}^*}]$ for $d \in \{0, 1\}$:

  return $\perp$

// Trivial win via explicit linking (1): $\mathcal{A}$ queried SLINK with challenge sigs, or sigs in different clusters

if $\exists \boldsymbol{\Sigma}_o$ s.t. $(*, \boldsymbol{\Sigma}_o) \in \mathsf{LNK}[\mathsf{uid}_d^*] \wedge$

  $(\boldsymbol{\Sigma}_o \cap \mathsf{CSIG} \neq \emptyset \vee \boldsymbol{\Sigma}_o \notin \mathsf{SIG}[\mathsf{uid}_d^*] \vee \nexists i_{\mathsf{SIG}^*}$ s.t. $\boldsymbol{\Sigma}_o \in \mathsf{SIG}^*[\mathsf{uid}_d^*, i_{\mathsf{SIG}^*}])$ for $d \in \{0, 1\}$:

  return $\perp$

// Trivial win via explicit linking (2): $\mathcal{A}$ queried $\mathsf{CH\text{-}SSIGN}_b$ with challenge sigs in different clusters

if $\exists \boldsymbol{\Sigma}_o$ s.t. $(*, \boldsymbol{\Sigma}_o) \in \mathsf{CLNK} \wedge \nexists i_{\mathsf{CSIG}}$ s.t. $\boldsymbol{\Sigma}_o \in \mathsf{CSIG}[i_{\mathsf{CSIG}}]$:

  return $\perp$

return b'

4.2 Sequential Construction

We describe how we add such sequential behaviour to Π_{UCL} while preserving the desired anonymity. Recall that signatures must remain unlinkable and *not* reveal user-specific order (such as being the 5-th signature of some user). The order is only guaranteed and re-established for the subset of signatures linked via SLink.

Adding order information. Our construction leverages well known hash-chain structures [26]. Roughly, every i-th signature is extended with information linking it to the $(i-1)$-th signature by the same user. For this, we use pseudorandom numbers. First, x_i is generated for the i-th signature, and combined with x_{i-1}, from the previous signature, by computing $\mathsf{H}(x_i \oplus x_{i-1})$. The result of this hash and $\mathsf{H}(x_i)$ are added to the signature. In sequential link proofs, besides the basic link proof, the signer reveals the x_i's of all the signatures in the sequence.

Trusting an append-only bulletin board BB. In our sequential scheme construction, the BB is *required*. It now also checks that the commitments to the pseudorandom numbers specified above are unique across all the uploaded signatures: this is critical to prevent malleable sequences. Also, being *append-only* prevents removing signatures once added, avoiding tampering with order.

Our construction Π_{sUCL}. For brevity, we only describe the modified functions.

$\langle \mathsf{Join}(ipk), \mathsf{Issue}(ipk, isk)\rangle \rightarrow ((usk, st), \perp)$. Operates as in Π_{UCL}, but the user adds $k \leftarrow \mathsf{PRF.KeyGen}(\tau)$ to her usk and sets $st \leftarrow 1$.

$\mathsf{SSign}(ipk, usk, st, m, scp) \rightarrow ((\tilde{\sigma}, nym), st')$. Computes (σ, nym) as in $\Pi_{\mathsf{UCL}}.\mathsf{Sign}$ and extends σ with the anonymous sequence seq using the key k and state st:

- Parse usk as (A, x, y, s, k) and compute (σ, nym) as in Sign.
- Compute $n_{st} \leftarrow \mathsf{PRF.Eval}(k, 0||st)$, $n_{st-1} \leftarrow \mathsf{PRF.Eval}(k, 0||st-1)$.
- Compute $x_{st} \leftarrow \mathsf{PRF.Eval}(k, 1||n_{st})$, $x_{st-1} \leftarrow \mathsf{PRF.Eval}(k, 1||n_{st-1})$.
- Compute $seq_1 \leftarrow \mathsf{H}'(x_{st}), seq_2 \leftarrow \mathsf{H}'(x_{st} \oplus x_{st-1}), seq_3 \leftarrow n_{st}$.
- Set $seq \leftarrow (seq_1, seq_2, seq_3)$, $st \leftarrow st + 1$.
- Return $(((\sigma, seq), nym), st)$.

The signatures in our construction are required to be uploaded to the bulletin board BB. The entity responsible to do so may depend on the use case. BB verifies $(m, scp, (\sigma, (seq_1, seq_2, seq_3)), nym)$ and checks uniqueness of seq, rejecting the signature if either check fails. Uniqueness of seq ensures that no $\Sigma' = (\cdot, \cdot, (\cdot, (seq'_1, seq'_2, \cdot)), \cdot)$ exists in BB, such that $seq_1 = seq'_1$ or $seq_2 = seq'_2$.

$\mathsf{SLink}(ipk, usk, lm, \boldsymbol{\Sigma}_o) \rightarrow \pi_{seq}/\perp$. Sequential link proofs are computed as previous link proofs, but adding to the proof the commitment openings. Namely:

- Parse usk as (A, x, y, s, k) and $\boldsymbol{\Sigma}_o$ as $\{\Sigma_i = (\cdot, \cdot, (\cdot, (\cdot, \cdot, seq_{i,3})), \cdot)\}_{i\in[n]}$
- If any Σ_i does not exist in BB, abort. Else, compute π_l as in Link.
- For all Σ_i in $\boldsymbol{\Sigma}_o$, compute $x_i \leftarrow \mathsf{PRF.Eval}(k, 1||seq_{i,3})$.
- Return $\pi_{seq} \leftarrow (\pi_l, \{x_i\}_{i\in[n]})$.

$\mathsf{VerifySLink}(ipk, lm, \boldsymbol{\Sigma}_o, \pi_{seq}) \rightarrow 0/1$. Verifiers check the link proof as in the basic scheme, and recompute and compare the hash-chain:

- Parse π_{seq} as $(\pi_l, \{x_i\}_{i\in[n]})$, and $\boldsymbol{\Sigma}_o$ as $\{\Sigma_i = (\cdot, \cdot, (\cdot, (seq_{i,1}, seq_{i,2}, \cdot)), \cdot)\}_{i\in[n]}$.
- If any Σ_i does not exist in BB, return 0. Else, verify π_l as in $\mathsf{VerifyLink}$.
- Check $seq_{1,1} = \mathsf{H}'(x_1)$. If not, reject.
- For $i \in [2, n]$, check $seq_{i,1} = \mathsf{H}'(x_i)$ and $seq_{i,2} = \mathsf{H}'(x_i \oplus x_{i-1})$. If not, reject.

Efficiently fetching previously created signatures. Finally, note that users can leverage the n_{st} values to easily fetch signatures from the bulletin board BB. If a user has a rough idea of the value of st when the signature was created, she can use PRF to recompute n_{st} for near st values. Otherwise, it is always possible to iterate from the initial value until finding the desired signature (as opposed to locally storing all signatures, or iterating through all signatures in BB).

4.2.1 Security of our Construction

Theorem 2. *Assuming zero-knowledgeness and simulation-soundness of* SPK*, collision resistance of* H'*, pseudorandomness of* PRF*, and a trusted* BB *verifying signatures and checking uniqueness of seq (across all signatures in* BB*), our construction is secure under the discrete logarithm, DDH, and q-SDH assumptions, in the random oracle model for* H*,* H' *and* SPK*.*

Proof sketch. Proving *anonymity* essentially requires showing that the newly added *seq* components can be simulated, which follows from pseudorandomness of PRF and the modelling of H and H' as random oracles.

For *sequentiality*, we show how to find collisions in H', assuming a trusted BB verifying signatures and checking uniqueness of their *seq* components, and pseudorandomness of PRF. Since honest signatures must exist in $\boldsymbol{\Sigma}^*$, all the attacker can do is to remove or swap honest signatures, or insert dishonest signatures before or after honest ones. However, the adversary commits to the set $\boldsymbol{\Sigma}'$ of dishonest signatures in the first stage of the game, and he can only use signatures in this set and $\mathsf{SIG}[\mathsf{uid}^*]$ to produce $\boldsymbol{\Sigma}^*$. First, the uniqueness checks by BB prevent the adversary from creating multiple signatures with the same *seq* values and re-order them as desired. Then, we show that to remove or swap honest signatures, or insert malicious ones, the adversary must find different openings to the seq_1 or seq_2 values in the commited signatures that are consistent with their hash chain, implying a collision in H'. This ensures that, before corrupting the user, the probability of the adversary producing a dishonest signature that can be "chained" with an honest one, is negligible.

Full proofs for the new and modified properties are given in Appendix C. The rest of the properties are proven as in the basic scheme.

5 Evaluation and Measurements

Table 3 summarises the functionality provided by the UCL and sUCL variants proposed in the present work, as well as that of the most related works [22,25]. The table focuses on the linkability aspects, and on which are the entities that can perform such linking.

We now analyse the computational and space costs of our constructions, comparing with related work. In Table 4, we denote with $\mathsf{e}_{\mathbb{G}_X}$, p and h, respectively, an exponentiation (in $\mathbb{G}_X$), a pairing and a computation of a hash function; and with $n\mathbb{G}_1$, $n\mathbb{Z}_p$, $n\mathsf{h}$, n elements in $\mathbb{G}_1$, $\mathbb{Z}_p$ and hashes, respectively (also, elements associated to the Paillier encryption used in [22] are denoted with $\mathbb{Z}_{n^2}$).

	User-controlled Linking	Authority-controlled Linking	Sequential Proofs
UCL (Section 3)	Yes	No	No
sUCL (Section 4)	Yes	No	Yes
GL19 [22]	No	Yes	No
KSS19 [25]	Yes	Yes	No

Table 3: Functionality comparison between the schemes presented here and [22,25].

For the SPKs, we use the Fiat-Shamir transform, and for the PRF an HMAC construction [4]. The used curve is BLS12-381 [3,2]. The costs derived from verifying and storing the individual signatures involved in Link and VerifyLink are omitted, i.e., we only account for the costs derived from storing/computing or verifying the linkability proof itself. Note also that [22] does not include a linking functionality per se. The (mostly) equivalent functionality is a combination of their `Blind`, `Convert` and `Unblind` operations. Thus, in the table we show the aggregate of their costs. In addition, other operations supported by [25], but not compatible with our model, are also omitted. These include their `Opn`, `Lnk` and `LnkJdg` functions (in Table 4, `Link` and `VerifyLink` refers to `SLnk` and `SLnkJdg` in [25]).

Algorithm	**Our scheme**	KSS19 [25]	GL19 [22]
Join	$3\mathsf{p} + 1\mathsf{e}_{\mathbb{G}_T} + 3\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{h}$	$8\mathsf{p}$	$3\mathsf{p} + 1\mathsf{e}_{\mathbb{G}_T} + 3\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{h}$
Issue	$4\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{h}$	$6\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{e}_{\mathbb{G}_2}$	$4\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{h}$
SSign	$14\mathsf{e}_{\mathbb{G}_1} + 2\mathsf{h} + 10\mathsf{h}$	$9\mathsf{p} + 13\mathsf{e}_{\mathbb{G}_1} + 6\mathsf{e}_{\mathbb{G}_T} + 2\mathsf{h}$	$16\mathsf{e}_{\mathbb{G}_1} + 15\mathsf{e}_{\mathbb{Z}_{n^2}} + 1\mathsf{h}$
Verify	$2\mathsf{p} + 9\mathsf{e}_{\mathbb{G}_1} + 2\mathsf{h}$	$9\mathsf{p} + 12\mathsf{e}_{\mathbb{G}_1} + 7\mathsf{e}_{\mathbb{G}_T} + 2\mathsf{h}$	$2\mathsf{p} + 12\mathsf{e}_{\mathbb{G}_1} + 11\mathsf{e}_{\mathbb{Z}_{n^2}} + 1\mathsf{h}$
SLink (s sigs.)	$(s+1)\mathsf{h} + (s+2)\mathsf{e}_{\mathbb{G}_1} + 2s\mathsf{h}$	$2s\mathsf{e}_{\mathbb{G}_1} + (s+1)\mathsf{h}$	$(7s+8)\mathsf{e}_{\mathbb{G}_1}$
VerifySLink (s sigs.)	$(s+1)\mathsf{h} + 2\mathsf{e}_{\mathbb{G}_1} + (2s-1)\mathsf{h}$	$2s\mathsf{e}_{\mathbb{G}_1} + 1\mathsf{h}$	N/A

	Our scheme	KSS19 [25]	GL19 [22]
Signature	$4\mathbb{G}_1 + 1\mathsf{H} + 5\mathbb{Z}_p + 3\mathsf{H}$	$6\mathbb{G}_1 + 1H + 5\mathbb{Z}_p$	$3\mathbb{G}_1 + 6\mathbb{Z}_p + 1\mathsf{H} + 6\mathbb{Z}^*_{n^2}$
Linkability Proof (s sigs.)	$1\mathsf{H} + 1\mathbb{Z}_p + s\mathbb{Z}_p$	$1H + s\mathbb{Z}_p$	N/A

Table 4: Computational (top) and space (bottom) costs. In the "Our scheme" column, we show in black font the costs of the UCL scheme (Section 3), and the text in red corresponds to the added costs of the sUCL scheme (Section 4). Since [25,22] only support explicit linkability, we only compare the linking costs in those schemes against the explicit linking of our schemes. Link costs for [22] aggregate their blinding, converting and unblinding costs. Operations from [25] that are not compatible with our model are omitted.

Fig. 6 shows the results of experiments obtained with a C implementation of both variants of our scheme (run on a MacBook Pro 2.5 GHz Quad-Core Intel i7, 16 GB 2133 MHz LPDDR3 RAM), and iterating every trial 1000 times. Setup, Join and Issue are omitted, as they will typically take place either rarely or in non time-critical contexts. Sign and Verify run in well below 5ms. For Link and VerifyLink (and the sequential variants), we experiment with sets of 10, 50 and 100 signatures. As in Table 4, this does not include verification of individual signatures. Note that even in the case of 100 signatures, we are still in the order of 40ms for linking and 20ms for verifying the proofs. For comparison, [25] reports signing and signature verification times around 100-150 ms, and linking and link verification times (for only two signatures) in the order of 330 ms.

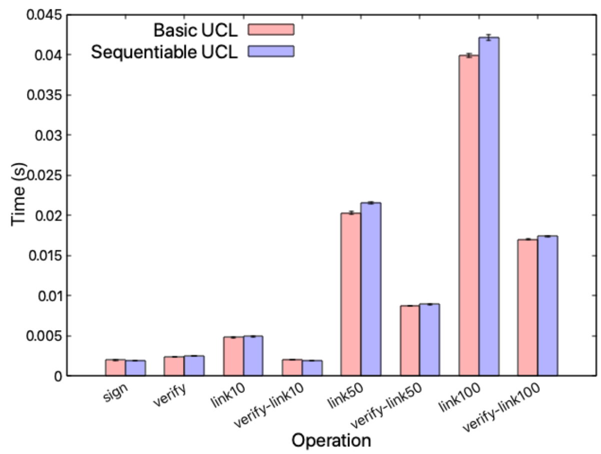


Fig. 6: Costs for Sign, Verify and Link (with 10, 50 and 100 signatures).

6 Conclusion

We have presented a new variant of group signatures that allows users to explicitly link large sets of signatures, supports implicit signature linking, and does not rely on a trusted opener. We have then extended this to allow proving order within a sequence of linked signatures, including that no signature has been omitted which was originally produced between the first and last signatures of the sequence. We have also given a formal model capturing the extended unforgeability and privacy properties in this setting, and efficient constructions realizing our model, which we have proved secure under discrete logarithm related assumptions. We have also reported on experimental evaluation obtained from an implementation of our schemes.

Several lines of further work are possible. First, we give an unforgeability property ensuring that order is maintained against honest-then-corrupt users, but we do not consider the equivalent for initially corrupt ones. While we argue that modelling honest-then-corrupt users is applicable to many real-world use cases, it is interesting to consider the stronger variant. In that case, initially, it seems that we can only hope to detect inconsistent proofs. Otherwise, if we only consider independent sequence proofs, a malicious signer may just "precompute" the sequence in the order he intends to prove afterwards, even if he publishes the signatures in a different order. Also, being able to prove non-linkage of signatures may be an interesting functionality – which would also impact the model. In practice, there may be use cases where proving not having issued a (set of) signature(s) can be useful. For instance, as a basic mechanism for (privacy respectful) blacklisting. Efficiency-wise, taking inspiration on [19,24], a great improvement would be to study the incorporation of batch verification of signatures (in addition to batch linking). On a more specific note, our construction for proving linked sequences introduces an artifact that affects the anonymity property. Namely, separately linking two adjacent sequences (i.e., where the last signature of one sequence was created immediately before the first signature of the other) makes both sequences linkable. Hence, removing this constraint would be an obvious improvement.

Acknowledgements. This work has been supported by the European Union's Horizon 2020 research and innovation program under Grant Agreement Number 768953 (ICT4CART). Jesus thanks Patrick Towa for very insightful and helpful discussions. Part of Anja's work was done while she was at IBM Research – Zurich.

References

1. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-taa. In: Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings. pp. 111–125 (2006)
2. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. J. Cryptology **32**(4), 1298–1336 (2019)
3. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: SCN 2002. pp. 257–267 (2002)
4. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO '96, Proceedings. pp. 1–15 (1996)
5. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: CT-RSA 2005, Proceedings. pp. 136–153 (2005)
6. Bernhard, D., Fuchsbauer, G., Ghadafi, E., Smart, N.P., Warinschi, B.: Anonymous attestation with user-controlled linkability. Int. J. Inf. Sec. **12**(3), 219–249 (2013). https://doi.org/10.1007/s10207-013-0191-z, `https://doi.org/10.1007/s10207-013-0191-z`
7. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: ASIACRYPT 2012. Proceedings. pp. 626–643 (2012)
8. Bichsel, P., Camenisch, J., Neven, G., Smart, N.P., Warinschi, B.: Get shorty via group signatures without encryption. In: Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings. pp. 381–398 (2010)

9. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptology **21**(2), 149–177 (2008)
10. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: ACM CCS 2004. pp. 168–177 (2004)
11. Camenisch, J., Drijvers, M., Lehmann, A.: Anonymous attestation using the strong diffie hellman assumption revisited. In: TRUST 2016, Proceedings. pp. 1–20 (2016)
12. Camenisch, J., Drijvers, M., Lehmann, A.: Universally composable direct anonymous attestation. In: PKC 2016, Proceedings. pp. 234–264 (2016)
13. Camenisch, J., Drijvers, M., Lehmann, A., Neven, G., Towa, P.: Short threshold dynamic group signatures. IACR Cryptology ePrint Archive **2020**, 16 (2020)
14. Camenisch, J., Krenn, S., Lehmann, A., Mikkelsen, G.L., Neven, G., Pedersen, M.Ø.: Formal treatment of privacy-enhancing credential systems. In: SAC 2015, Revised Selected Papers. pp. 3–24 (2015)
15. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: EUROCRYPT 2001, Proceeding. pp. 93–118 (2001)
16. Cash, D., Kiltz, E., Shoup, V.: The twin diffie-hellman problem and applications. In: Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. pp. 127–145 (2008)
17. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT '91, Proceedings. pp. 257–265. Springer (1991)
18. Choi, S.G., Park, K., Yung, M.: Short traceable signatures based on bilinear pairings. In: IWSEC 2006, Proceedings. pp. 88–103 (2006)
19. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Fischlin, M. (ed.) Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5473, pp. 309–324. Springer (2009). https://doi.org/10.1007/978-3-642-00862-7_21, `https://doi.org/10.1007/978-3-642-00862-7_21`
20. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO '86, Proceedings. pp. 186–194 (1986)
21. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008)
22. Garms, L., Lehmann, A.: Group signatures with selective linkability. In: PKC 2019, Proceedings. pp. 190–220 (2019)
23. Kiayias, A., Tsiounis, Y., Yung, M.: Traceable signatures. In: EUROCRYPT 2004, Proceedings. pp. 571–589 (2004)
24. Kim, H., Lee, Y., Abdalla, M., Park, J.H.: Practical dynamic group signature with efficient concurrent joins and batch verifications. Cryptology ePrint Archive, Report 2020/921 (2020), `https://eprint.iacr.org/2020/921`
25. Krenn, S., Samelin, K., Striecks, C.: Practical group-signatures with privacy-friendly openings. In: ARES 2019. pp. 10:1–10:10 (2019)
26. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: CCS 2017. pp. 455–471 (2017)
27. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004)
28. Song, D.X.: Practical forward secure group signature schemes. In: CCS 2001, Proceedings. pp. 225–234 (2001)

29. Young, A.L., Yung, M.: Semantically secure anonymity: Foundations of re-encryption. In: Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings. pp. 255–273 (2018)

A Implementation Notes

We have implemented the basic and sequential instantiations of our scheme. The efficiency analysis presented in Section 5 is based on that implementation, which is available at `https://github.com/IBM/libgroupsig`. Additionally, we have prepared a demo web application that leverages our implementation. It can be accessed from any PC with access to the Internet and a local installation of Docker[3] via the following commands:

```
$ docker pull jdiazvico/sucl:latest
$ docker run -p 5000:5000 jdiazvico/sucl
```

Where the first command downloads a Docker image that has a local installation of the compiled code, and the second command runs the demo. After successfully running both commands, the demo – which contains explanatory instructions on how to use it – can be accessed by going to `http://127.0.0.1:5000` on any web browser.

B Proofs of Security for Π_{UCL}

B.1 Correctness

– **Correctness of** Sign**.** We formalize this property with the game in Fig. 7. The adversary wins if it outputs an (m, scp) pair for which an honestly generated signature does not verify.

Experiment: $\mathbf{Exp}^{\mathsf{corr\text{-}sign}}_{\mathcal{A},\mathsf{UCL}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau)$
$(ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}, m, scp) \leftarrow \mathcal{A}^{\mathsf{ADDU}}(ipk)$
$(\sigma, nym) \leftarrow \mathsf{Sign}(ipk, \mathsf{USK}[\mathsf{uid}], m, scp)$
if $\mathsf{Verify}(ipk, m, scp, \sigma, nym) = 0$: **return** 1
return 0

Fig. 7: Correctness of sign experiment.

[3] `https://www.docker.com/`. Last access on October 10th, 2020.

Proof. A signature produced by an honest user over a message m with scope scp is a tuple (σ, nym), such that $nym = \mathsf{H}(scp)^y$ and $\sigma = (A', \hat{A}, d, \pi_\sigma)$, where $\pi_\sigma = \mathsf{SPK}\{(x, y, r_2, r_3, s') : nym = \mathsf{H}(scp)^y \wedge \hat{A}/d = (A')^{-x} h_2^{r_2} \wedge g_1 h_1^y = d^{r_3} h_2^{-s'}\}(ctx, m)$. The proof π_σ is correct due to the correctness of the underlying SPK. Additionally, $e(A', ipk) = e(\hat{A}, g_2)$, since $\hat{A} = (A)^{-xr_1}(g_1 h_1^y h_2^s)^{r_1} = (g_1 h_1^y h_2^s)^{\frac{r_1 isk}{isk+x}} = (A')^{isk}$ and $A' \neq 1_{\mathbb{G}_1}$ with overwhelming probability because $A \neq 1_{\mathbb{G}_1}$. Therefore, honestly generated signatures verify correctly.

- **Correctness of** Link. This property is formalized in Fig. 8. The adversary wins if it returns a set of (m_i, scp_i) pairs for which a matching set of honestly generated signatures and an honestly generated proof of linking π_l do not verify.

Experiment: $\mathbf{Exp}^{\mathsf{corr\text{-}link}}_{\mathcal{A},\mathsf{UCL}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau)$
$(ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}, \{m_i, scp_i\}_{i\in[n]}, lm) \leftarrow \mathcal{A}^{\mathsf{ADDU}}(ipk)$
for $i \in [n] : (\sigma_i, nym_i) \leftarrow \mathsf{Sign}(ipk, \mathsf{USK}[\mathsf{uid}], m_i, scp_i)$
$\pi_l \leftarrow \mathsf{Link}(ipk, \mathsf{USK}[\mathsf{uid}], m, \{(m_i, scp_i, \sigma_i, nym_i)\}_{i\in[n]})$
if $\mathsf{VerifyLink}(ipk, m, \{(m_i, scp_i, \sigma_i, nym_i)\}_{i\in[n]}, \pi_l) = 0$:
  return 1
return 0

Fig. 8: Correctness of link experiment.

Proof. Let (A, x, y, s) be the secret key of user uid, and $\{(m_i, scp_i, \sigma_i, nym_i)\}_{i\in[n]}$ and π_l respectively be a set of signed messages and proof of them being linked, all created honestly by uid. By correctness of the SPK in each signature, we have that $nym_i = \mathsf{H}(scp_i)^y$. Thus, we have at $\mathsf{VerifyLink}$ that $\overline{nym} = \prod_{i\in[n]} nym_i = \prod_{i\in[n]} \mathsf{H}(scp_i)^y = (\prod_{i\in[n]} \mathsf{H}(scp))^y = \overline{hscp}^y$ and by the correctness of the SPK in the honestly generated π_l, it verifies correctly.

B.2 Anonymity

Anonymity follows from the zero-knowledge and simulation-soundness properties of the SPKs, and the fact that pseudonyms are indistinguishable from random when different scopes are used, under the DDH assumption.

Proof (Anonymity). We build an adversary $\mathcal{A}'$ who, given an adversary $\mathcal{A}$ that wins the anonymity game, uses it to break the DDH assumption. Next, we argue how does $\mathcal{A}'$ simulate the inputs to $\mathcal{A}$ and show the reduction to DDH.

Adversary $\mathcal{A}'$. $\mathcal{A}'$ leverages DDH's random self-reducibility property to produce t DDH tuples of the shape $(\tilde{\mathsf{D}}_0 = g, \tilde{\mathsf{D}}_i = g^{\tilde{\mathsf{a}}_i}, \tilde{\mathsf{D}}_{t+1} = g^{\tilde{\mathsf{b}}}, \tilde{\mathsf{D}}_{t+i+1} = g^{\tilde{\mathsf{a}}_i\tilde{\mathsf{b}}})$[4]. It randomly picks a bit b and the honest user uid^* in which to embed the challenge among the q total honest users added by $\mathcal{A}$, and answers $\mathcal{A}$ oracle queries (up to t $\mathsf{CH\text{-}SIGN}_b$ and SIGN queries for uid^*, where t can be made arbitrarily large). If $\mathcal{A}$ guesses b correctly, it returns 1; otherwise, returns 0. If the uid^* guessed at random does not match uid_b^* as returned by $\mathcal{A}$, $\mathcal{A}'$ exits early returning a random bit b'. $\mathcal{A}'$ is fully defined in Fig. 9. The oracles used by $\mathcal{A}'$ are defined in Fig. 10.

$\mathcal{A}'(\mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}}, \mathsf{D}_2 = g^{\mathsf{b}}, \mathsf{D}_3 = g^{\mathsf{c}})$

$\tilde{\mathsf{D}}_0, ..., \tilde{\mathsf{D}}_{2t+1} \leftarrow \mathsf{DDHRerand}_t(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$
$b, b' \leftarrow_\$ \{0,1\}, \mathsf{HL} \leftarrow \varnothing, i_{\mathsf{uid}} \leftarrow 0, i_{\mathsf{HL}} \leftarrow 0, k_{\mathsf{uid}} \leftarrow_\$ [q]$
$h_1 \leftarrow \tilde{\mathsf{D}}_0$, Generate remaining $param$ with $\mathsf{Setup}(1^\tau)$
$(ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}_0^*, \mathsf{uid}_1^*, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{LINK}}(\text{choose}, ipk, isk)$
if $\mathsf{uid}_b^* \neq \mathsf{uid}^*$: **return** b'
if $\mathsf{USK}[\mathsf{uid}_d^*] \neq \perp$ for $d = 0, 1$:
  Initialize $\mathsf{CH\text{-}SIGN}_b$ and $\mathsf{CH\text{-}LINK}_b$ with uid_b^*
else : **return** b'
$\mathsf{b}^* \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CH\text{-}SIGN}_b,\mathsf{LINK},\mathsf{CH\text{-}LINK}_b}(\text{guess}, \mathsf{state})$
if any oracle call returns $\perp$: **return** b'
if $\mathsf{b}^* = b$: **return** 1
return 0

$\mathsf{DDHRerand}_t(p, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$

$u_1, ..., u_t, v, w_1, ..., w_t \leftarrow_\$ \mathbb{Z}_p^*$
$\tilde{\mathsf{D}}_0 \leftarrow \mathsf{D}_0, \tilde{\mathsf{D}}_{t+1} \leftarrow \mathsf{D}_2\mathsf{D}_0^v$
for $i \in [1, t]$:
  $\tilde{\mathsf{D}}_i \leftarrow \mathsf{D}_1^{u_i}\mathsf{D}_0^{w_i}$
  $\tilde{\mathsf{D}}_{t+i+1} \leftarrow \mathsf{D}_3^{u_i}\mathsf{D}_2^{w_i}\mathsf{D}_1^{u_i v}\mathsf{D}_0^{w_i v}$
return $(\tilde{\mathsf{D}}_0, ..., \tilde{\mathsf{D}}_{2t+1})$

Fig. 9: Adversary $\mathcal{A}'$ against DDH from $\mathcal{A}$ breaking anonymity.

Random Oracle (H *function).* The hash function, idealized as a random oracle, returns elements in $\mathbb{G}_1$. For newly queried values, the random oracle increments the i_{HL} counter and returns the DDH challenge element $\tilde{\mathsf{D}}_{i_{\mathsf{HL}}} = g^{\tilde{\mathsf{a}}_{i_{\mathsf{HL}}}}$, updating HL with the used index. For already queried values, the random oracle just browses HL and returns the corresponding element. Note that, in all cases, the output of the random oracle is a new (or, rather, unused) uniformly random value.

Simulating the SNDU *oracle.* The SNDU oracle used by $\mathcal{A}'$ operates in the exact same way than the default SNDU oracle except for the k_{uid}-th query. In that case, SNDU sets uid^*, assigns $\tilde{\mathsf{D}}_{t+1} = g^{\tilde{\mathsf{b}}}$ to the new user's Y value, and simulates π_Y. Due to the zero knowledge property of the proof system, the output produced by SNDU is indistinguishable from the output of the default SNDU oracle.

[4] To do it, we use the $\mathsf{DDHRerand}_t$ algorithm, which is a simple extension of the Expanded DDH Self-Reduction algorithm in [29]. See Appendix D for details. For simplicity, we have $\mathcal{A}'$ generate an arbitrary number t of such tuples. However, note that they can also be generated on demand, e.g., when H is queried.

Simulating the SIGN *oracle.* The signing oracle is exactly the same as the default SIGN oracle except for calls involving uid^*. For requests with $\mathsf{uid} = \mathsf{uid}^*$, $\mathcal{A}'$ fetches from HL the j value from the entry corresponding to scp and sets nym to the $\tilde{\mathsf{D}}_{t+j+1}$ element from the DDH tuple. Then, simulates the signature and π_σ, updates SIG and returns. Lets assume that $\mathcal{A}'$ receives a DDH tuple. Then, $nym = \tilde{\mathsf{D}}_{t+j+1} = g^{\tilde{\mathsf{a}}_j \tilde{\mathsf{b}}}$, which is consistent with SNDU, where we set $Y = \tilde{\mathsf{D}}_{t+1} = g^{\tilde{\mathsf{b}}}$, and with H, where we set $\mathsf{H}(scp) = \tilde{\mathsf{D}}_{j+1} = g^{\tilde{\mathsf{a}}_j}$. The BBS+ signature can be simulated, as well as π_σ, due to the zero-knowledge property of SPK. If the input received by $\mathcal{A}'$ is not a DDH tuple, then $nym = \tilde{\mathsf{D}}_{t+j+1} = g^{\tilde{\mathsf{c}}_j}$, $Y = g^{\tilde{\mathsf{b}}}$ and $\mathsf{H}(scp) = g^{\tilde{\mathsf{a}}_j}$. In this case, both the zero-knowledge and simulation soundness properties of SPK ensure that the BBS+ signature and π_σ can be simulated. Since nyms in real executions are indistinguishable from random [27], simulated nyms when the input is not a DDH tuple are indistinguishable from real nyms. Consequently, in all cases, the output produced by SIGN in the simulation is indistinguishable from the output of the default SIGN oracle. Moreover, when the input is not a DDH tuple, the output is completely independent from the bit b.

Simulating the LINK *oracle.* Queries of $\mathsf{uid} \neq \mathsf{uid}^*$ are handled as in the default LINK oracle. When $\mathsf{uid} = \mathsf{uid}^*$, the $\overline{hscp}$ value is computed as usual, but in order to compute the $\overline{nym}$ value, we fetch from HL the (scp_i, j) entry for each scope in the set of signatures, and let $\overline{nym} = \prod_j \tilde{\mathsf{D}}_{t+j+1}$. If the input to $\mathcal{A}'$ is a DDH tuple, then $\overline{nym} = g^{\tilde{\mathsf{b}}(\sum_j \tilde{\mathsf{a}}_j)}$, which is exactly the value of $\overline{nym}$ in real executions, and π_l can be simulated from $\overline{hscp}$ and $\overline{nym}$ due to the zero-knowledge property of SPK. If the input to $\mathcal{A}'$ is not an DDH tuple, then $\overline{nym} = g^{\sum_j \tilde{\mathsf{c}}_j}$. In this case, the zero-knowledge and simulation soundness properties of SPK ensure that π_l can be simulated. Finally, note that in both cases, the output of LINK is indistinguishable from the default LINK oracle and, additionally, when a DDH tuple is given to $\mathcal{A}'$, the behaviour of the oracle is the same as with the default LINK oracle; while when a DDH tuple is not input, the output is completely independent from the bit b.

Simulating the $\mathsf{CH\text{-}SIGN}_b$ *oracle.* The argument made for the simulation of the SIGN oracle holds for $\mathsf{CH\text{-}SIGN}_b$ as well. Consequently, in all cases, the output produced by $\mathsf{CH\text{-}SIGN}_b$ is indistinguishable from the output of the default $\mathsf{CH\text{-}SIGN}_b$ oracle. Moreover, when the input is not a DDH tuple, the output of the $\mathsf{CH\text{-}SIGN}_b$ oracle is completely independent from the bit b.

Simulating the $\mathsf{CH\text{-}LINK}_b$ *oracle.* The argument made for the simulation of LINK when $\mathsf{uid} = \mathsf{uid}^*$ also holds for $\mathsf{CH\text{-}LINK}_b$. Thus, the output of simulations of $\mathsf{CH\text{-}LINK}_b$ is indistinguishable from real executions and, when the input of $\mathcal{A}'$ is not a DDH tuple, the ouptut is completely independent from the bit b.

Reduction to DDH. From the oracles' description, we can see that all information $\mathcal{A}$ receives is exactly the same it receives in the default anonymity game when a DDH tuple is given to $\mathcal{A}'$; and it is independent from the challenge bit b when the input to $\mathcal{A}'$ is not a DDH tuple. However, we have to account for the probability that $\mathcal{A}'$ returns early due to $\mathsf{uid}_b^* \neq \mathsf{uid}^*$, case in which $\mathcal{A}'$ returns a bit chosen uniformly and random, and which happens with probability $1 - 1/q$, as-

Global variables (set by $\mathcal{A}'$)

$\mathsf{uid}^*, i_{\mathsf{HL}}, i_{\mathsf{uid}}, k_{\mathsf{uid}}$
$(\tilde{\mathsf{D}}_0 = g, \tilde{\mathsf{D}}_1 = g^{\tilde{a}_1}, \tilde{\mathsf{D}}_{t+1} = g^{\tilde{b}}, \tilde{\mathsf{D}}_{t+2} = g^{\tilde{c}_1})$
...
$(\tilde{\mathsf{D}}_0 = g, \tilde{\mathsf{D}}_t = g^{\tilde{a}_t}, \tilde{\mathsf{D}}_{t+1} = g^{\tilde{b}}, \tilde{\mathsf{D}}_{2t+1} = g^{\tilde{c}_t})$

$\mathsf{SNDU}(\mathsf{uid}, \mathsf{M}_{in})$

Check input as in default SNDU
Update HUL as in default SNDU
$i_{\mathsf{uid}} \leftarrow i_{\mathsf{uid}} + 1$, Parse M_{in} as n
if $i_{\mathsf{uid}} = k_{\mathsf{uid}}$:
  $\mathsf{uid}^* \leftarrow \mathsf{uid}, Y \leftarrow \tilde{\mathsf{D}}_{t+1}$, Simulate π_Y with Y, n
  return $((Y, \pi_Y), \mathsf{cont})$
Continue from line 6 of SNDU

$\mathsf{SIGN}(\mathsf{uid}, m, scp)$

Check input as in default SIGN
if $\mathsf{uid} = \mathsf{uid}^*$:
  Find $(scp, j) \in \mathsf{HL}$;
  $nym \leftarrow \tilde{\mathsf{D}}_{t+j+1}, A', d \leftarrow_{\$} \mathbb{Z}_p^*, \hat{A} \leftarrow (A')^{isk}$
  Simulate π_σ with $A', \hat{A}, d, scp, nym, m$
  $\sigma \leftarrow (A', \hat{A}, d, \pi_\sigma), \Sigma \leftarrow (m, scp, \sigma, nym)$
  $\mathsf{SIG} \leftarrow \mathsf{SIG} \cup \{(\mathsf{uid}, \Sigma)\}$
  return (σ, nym)
else : Run SIGN as usual

$\mathsf{CH\text{-}LINK}_b(lm, \boldsymbol{\Sigma})$

Check input as default $\mathsf{CH\text{-}LINK}_b$
$\mathsf{LNK} \leftarrow \mathsf{LNK} \cup \{(\mathsf{uid}, lm, \boldsymbol{\Sigma})$
Parse $\boldsymbol{\Sigma}$ as $\{(m_i, scp_i, \sigma_i, nym_i)\}_{i \in [n]}$
$\overline{hscp} \leftarrow \prod_{i \in [n]} \mathsf{H}(scp_i), \overline{nym} \leftarrow 1$
for $i \in [n]$: Find $(scp_i, j) \in \mathsf{HL}, \overline{nym} \leftarrow \overline{nym} \cdot \tilde{\mathsf{D}}_{t+j+1}$
Simulate π_l with $\overline{hscp}, \overline{nym}$
return π_l

$\mathsf{H}(\mathsf{in})$

if $(\mathsf{in}, i) \in \mathsf{HL}$: **return** $\tilde{\mathsf{D}}_i$
$i_{\mathsf{HL}} \leftarrow i_{\mathsf{HL}} + 1$
$\mathsf{HL} \leftarrow \mathsf{HL} \cup \{(\mathsf{in}, i_{\mathsf{HL}})\}$
return $\tilde{\mathsf{D}}_{i_{\mathsf{HL}}}$

$\mathsf{CH\text{-}SIGN}_b(m, scp)$

Check input as in default $\mathsf{CH\text{-}SIGN}_b$
Find $(scp, j) \in \mathsf{HL}$
$nym \leftarrow \tilde{\mathsf{D}}_{t+j+1}, A', d \leftarrow_{\$} \mathbb{Z}_p^*, \hat{A} \leftarrow (A')^{isk}$
Simulate π_σ with $A', \hat{A}, d, scp, nym, m$
$\sigma \leftarrow (A', \hat{A}, d, \pi_\sigma), \Sigma \leftarrow (m, scp, \sigma, nym)$
$\mathsf{CSIG} \leftarrow \mathsf{CSIG} \cup \{\Sigma\}$
return (σ, nym)

$\mathsf{LINK}(\mathsf{uid}, lm, \boldsymbol{\Sigma})$

Check input as default LINK
if $\mathsf{uid} = \mathsf{uid}^*$:
  $\mathsf{LNK} \leftarrow \mathsf{LNK} \cup \{(\mathsf{uid}, lm, \boldsymbol{\Sigma})$
  Parse $\boldsymbol{\Sigma}$ as $\{(m_i, scp_i, \sigma_i, nym_i)\}_{i \in [n]}$
  $\overline{hscp} \leftarrow \prod_{i \in [n]} \mathsf{H}(scp_i), \overline{nym} \leftarrow 1$
  for $i \in [n]$:
    Find $(scp_i, j) \in \mathsf{HL}, \overline{nym} \leftarrow \overline{nym} \cdot \tilde{\mathsf{D}}_{t+j+1}$
  Simulate π_l with $\overline{hscp}, \overline{nym}$
  return π_l
else : Run default LINK oracle

Fig. 10: Oracles used by $\mathcal{A}'$ against DDH in the anonymity game.

suming q executions of the SNDU oracle. Let S denote the event that $\mathcal{A}$ correctly guesses the bit b in the anonymity game. Then, under the DDH assumption, we have that $\mathbf{Adv}_{\mathcal{A}'}^{DDH}(\tau) = |\Pr[\mathbf{Exp}_{\mathcal{A}'}^{DDH-1}(\tau) = 1] - \Pr[\mathbf{Exp}_{\mathcal{A}'}^{DDH-0}(\tau) = 0]| = |(1 - \frac{1}{q})\frac{1}{2} + \frac{1}{q}\Pr[S] - (1 - \frac{1}{q})\frac{1}{2} - \frac{1}{q}\frac{1}{2}| = \frac{1}{q}|\Pr[S] - \frac{1}{2}| \leq \epsilon_{DDH}$. Therefore, $|\Pr[S] - \frac{1}{2}| \leq q\epsilon_{DDH}$, which is negligible under the DDH assumption.

B.3 Signature Traceability

Signature traceability follows from unforgeability of the BBS+ credentials (which holds under the q-SDH assumption), and zero-knowledgeness and soundness of SPK: the adversary cannot produce more signatures that corrupt users under his control, given the uniqueness of pseudonyms per scope.

Proof (Signature Traceability). We build an adversary $\mathcal{A}'$ who breaks the q-SDH assumption from adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ winning the signature non-frameability game through winning conditions 1 and 2, respectively.

First, we describe how does $\mathcal{A}'$ (against q-SDH) simulate inputs and oracle calls by adversaries $\mathcal{A}_1$ or $\mathcal{A}_2$. However, the core of the simulation is the same for both, so we use $\mathcal{A}_x$ instead of $\mathcal{A}_1$ or $\mathcal{A}_2$ in the definition of $\mathcal{A}'$. The difference in the reduction is highlighted after describing the oracles. $\mathcal{A}'$ is formally defined in Fig. 12 and its oracles in Fig. 13.

For this property, we need to define the $\mathsf{Identify}$ helper function used in the signature traceability game. We define it as in Fig. 11, which clearly meets the uniquness requirement Section 3.2, for an honest issuer [14]

$\mathsf{Identify}(ipk, \mathsf{uid}, \Sigma)$

---

if $\mathsf{uid} \notin \mathsf{HUL} \cup \mathsf{CUL}$: **return** $\perp$
if $\mathsf{uid} \in \mathsf{HUL}$: $usk \leftarrow \mathsf{USK}[\mathsf{uid}]$
else : Extract usk from $\mathsf{transcript}[\mathsf{uid}]$
Parse usk as (A, x, y, s)
if $\mathsf{Verify}(ipk, \Sigma) = 0$: **return** 0
Parse Σ as (m, scp, σ), σ as $(\cdot, \cdot, \cdot, \cdot, nym)$
if $nym = \mathsf{H}(scp)^y$: **return** 1
return 0

Fig. 11: Definition of $\mathsf{Identify}$ for the signature traceability reduction.

Adversary $\mathcal{A}'$. We follow the strategy in [12] to simulate up to q BBS+ signatures, which requires to randomly choose one (simulated) signature to be generated differently. For this, $\mathcal{A}'$ picks $k_{\mathsf{uid}} \leftarrow_\$ \mathbb{Z}_q$. With these simulated signatures, $\mathcal{A}'$ will be able to answer up to q queries by $\mathcal{A}_x$ to SNDI. Besides generating the simulated signatures, $\mathcal{A}'$ prepares the $param$ elements to pass to $\mathcal{A}_x$ in a consistent manner. Note that g_1 and g_2 are random elements in $\mathbb{G}_1$ and $\mathbb{G}_2$; h_2 is also a random generator of $\mathbb{G}_1$, since x_0, a, k are chosen uniformly at random; h_1 is a different random generator of $\mathbb{G}_2$, as μ is also chosen uniformly at random; and finally, ipk is also distributed as in the experiment, since $ipk = W = g_2^\chi$ for some unknown (random) $\chi \in \mathbb{Z}_p^*$.

Random Oracle (H function). The random oracle just computes a new random element in $\mathbb{G}_1$ and updates HL if the input was not queried before; or, if

$\underline{\mathcal{A}'(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{D}_0 = g_1^{\chi^0}, ..., \mathsf{D}_\mathsf{q} = g_1^{\chi^\mathsf{q}}, g_2, W = g_2^\chi)}$

$ipk \leftarrow W, k_\mathsf{uid} \leftarrow_\$ \mathbb{Z}_q, i \leftarrow 0$
$(d_1', g_1', \{B_i\}_{i\in[\mathsf{q}-1]}, \{x_i\}_{i\in[0,\mathsf{q}-1]})$
$\quad\quad \leftarrow \mathsf{GWeakBB}(\dots)$
$a, k, \mu \leftarrow_\$ \mathbb{Z}_p^*$
$h_2 \leftarrow ((d_1'(g_1')^{x_0})^k (g_1')^{-1})^{1/a}, h_1 \leftarrow h_2^\mu$
$(\Sigma_1, \dots, \Sigma_n) \leftarrow \mathcal{A}_x^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SIGN},\mathsf{LINK}}(ipk)$
Let $\Sigma_i = (m_i, scp, \sigma_i, nym_i)$ as in winning
  condition 1 or winning condition 2
Extract $\tilde{x}, \tilde{y}, r_2, r_3, s'$ from π_σ in σ_i
if $r_3 = 0 : \tilde{A} \leftarrow 1, s^* \leftarrow s', \tilde{s} \leftarrow s' + \mu\tilde{y}$
else :
  $\tilde{A} \leftarrow (A')^{r_3}, s^* \leftarrow s' + r_2 r_3 \tilde{s} \leftarrow s^* + \mu\tilde{y}$
return $((\tilde{A}(g_1')^{\frac{-k\tilde{s}}{a}})^{\frac{a}{a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0)}}, \tilde{x})$

$\underline{\mathsf{GWeakBB}(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{D}_0, \mathsf{D}_1, ..., \mathsf{D}_\mathsf{q}, g_2, W)}$

$\theta, x_0 \leftarrow_\$ \mathbb{Z}_p^*$
for $i \in [\mathsf{q}-1] : x_i \leftarrow_\$ \mathbb{Z}_p^*$
Define $f(X) = \prod_{i=1}^{\mathsf{q}-1}(X + x_i) =$
  $\sum_{i=0}^{\mathsf{q}-1} \alpha_i X^i, f_i(X) = f(X)/(X + x_i) = \sum_{j=0}^{\mathsf{q}-1} \beta_j X_j$
$g_1' \leftarrow \prod_{i=1}^{\mathsf{q}-1} \mathsf{D}_i^{\alpha_i\theta} = g_1^{\theta f(\chi)}, d_1' \leftarrow \prod_{i=0}^{\mathsf{q}-1} \mathsf{D}_{i+1}^{\alpha_i\theta} = (g_1')^\chi$
for $i \in [\mathsf{q}-1]$:
  $B_i \leftarrow \prod_{j=0}^{\mathsf{q}-1} \mathsf{D}_j^{\beta_j\theta} = g_1^{\theta f_i(\chi)} = (g_1')^{1/(\chi+x_i)}$
return $(d_1', g_1', \{B_i\}_{i\in[1,\mathsf{q}-1]}, \{x_i\}_{i\in[0,\mathsf{q}-1]})$

Fig. 12: Adversary $\mathcal{A}'$ against q-SDH from $\mathcal{A}_x$ breaking signature traceability ($x \in [1, 2]$ denotes the winning condition used by $\mathcal{A}_x$).

the input was queried before, returns the previously computed value. Thus, the outputs are uniformly random and consistent across queries.

Simulating the ADDU *oracle.* For queries to the ADDU oracle, $\mathcal{A}'$ just updates HUL and returns accept. The output of the ADDU oracle used by $\mathcal{A}'$ is trivially equally distributed to that of the ADDU oracle in the signature traceability experiment.

Simulating the SNDI *oracle.* $\mathcal{A}'$ first extracts y from π_Y. With the value of y, it already can complete the simulated BBS+ signature with the help of the previously computed weak BB08 signatures, accounting for the special case ($i = k_\mathsf{uid}$). If $i = k_\mathsf{uid}$, the result is a valid BBS+ signature, since $A_\mathsf{uid} = (g_1')^k = (g_1' h_2^{s_\mathsf{uid}} h_1^{y_\mathsf{uid}})^{\frac{1}{x_\mathsf{uid}+\chi}}$; also, x_uid and s_uid are random, as x_0 and a were chosen independently and uniformly at random. For all other queries, $\mathcal{A}'$ consumes a weak BB08 signature to build (up to $q - 1$) BBS+ additional signatures. Note also that in this case the signatures are correctly formed, since $A_\mathsf{uid} = B_i(B_i^{\frac{(x_0-x_i)k-1}{a}}(g_1')^{\frac{k}{a}})^{s_\mathsf{uid}+\mu y_\mathsf{uid}} = (g_1' h_2^{s_\mathsf{uid}} h_1^{y_\mathsf{uid}})^{\frac{1}{\chi+x_\mathsf{uid}}}$, and x_uid and s_uid are chosen independently and uniformly at random. Therefore, in all cases, the output of SNDI, as implemented by $\mathcal{A}'$, is indistinguishable to the output of the SNDI oracle in the signature traceability experiment.

Simulating the SIGN *oracle.* The SIGN oracle used by $\mathcal{A}'$ first computes a fresh y_uid if it was not already defined for the requested uid. Then, computes the nym as usual and simulates the BBS+ signature. Note that $\hat{A} = (A')^\chi$, as $d_1' = (g_1')^\chi$. Since r and d are chosen uniformly at random and independently, the produced signature is correctly formed and indistinguishable from signatures produced by the default SIGN oracle. Additionally, the zero-knowledge property of the underlying proof system ensures that π_σ can be indistinguishably simulated with respect to real signatures. Thus, outputs of the SIGN oracle used by $\mathcal{A}'$ are indistinguishable to outputs of the default SIGN oracle.

Simulating the LINK *oracle.* $\mathcal{A}'$ uses the same LINK oracle as in the signature traceability experiment.

Global variables (set by $\mathcal{A}'$)

$a, k, \mu, k_{\mathsf{uid}}, i$
$d'_1, g'_1, \{B_i\}_{i\in[\mathsf{q}-1]}, \{x_i\}_{i\in[0,\mathsf{q}-1]}$

$\mathsf{H}(\mathsf{in})$

if $(\mathsf{in}, R) \in \mathsf{HL}$: **return** R
$R \leftarrow_{\$} \mathbb{G}_1, \mathsf{HL} \leftarrow \mathsf{HL} \cup \{(\mathsf{in}, R)\}$
return R

$\mathsf{SNDI}(\mathsf{uid}, \mathsf{M}_{in})$

Check input as in default SNDI
Update $\mathsf{transcript}$, $\mathsf{dec}^{\mathsf{uid}}$ as in default SNDI
$i \leftarrow i + 1$; Extract y from π_Y
if $i = k_{\mathsf{uid}}$:
  $s_{\mathsf{uid}} \leftarrow a - \mu y, A_{\mathsf{uid}} \leftarrow (g'_1)^k$
  $x_{\mathsf{uid}} \leftarrow x_0, y_{\mathsf{uid}} \leftarrow y$
else :
  $x_{\mathsf{uid}} \leftarrow x_i, s_{\mathsf{uid}} \leftarrow_{\$} \mathbb{Z}_p^*, y_{\mathsf{uid}} \leftarrow y$
  $A_{\mathsf{uid}} \leftarrow B_i(B_i^{\frac{(x_0-x_i)k-1}{a}}(g'_1)^{\frac{k}{a}})^{s_{\mathsf{uid}}+\mu y_{\mathsf{uid}}}$
return $((A_{\mathsf{uid}}, x_{\mathsf{uid}}, s_{\mathsf{uid}}), \mathsf{cont})$

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

if $\mathsf{uid} \in \mathsf{HUL} \cup \mathsf{CUL}$: **return** $\perp$
else :
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{\mathsf{uid}\}$
  return accept

$\mathsf{SIGN}(\mathsf{uid}, m, scp)$

Check input as in default SIGN
if y_{uid} is undefined :
  $y_{\mathsf{uid}} \leftarrow_{\$} \mathbb{Z}_p^*, \mathsf{USK}[\mathsf{uid}] \leftarrow (\cdot, \cdot, y_{\mathsf{uid}}, \cdot)$
$nym \leftarrow \mathsf{H}(scp)^{y_{\mathsf{uid}}}, r \leftarrow_{\$} \mathbb{Z}_p^*,$
$A' \leftarrow (g'_1)^r, \hat{A} \leftarrow (d'_1)^r, d \leftarrow_{\$} \mathbb{Z}_p^*$
Simulate π using $A', \hat{A}, d, nym, m$
$\sigma \leftarrow (A', \hat{A}, d, \pi_\sigma), \Sigma \leftarrow (m, scp, \sigma, nym)$
$\mathsf{SIG} \leftarrow \mathsf{SIG} \cup \{(\mathsf{uid}, \Sigma)\}$
return (σ, nym)

Fig. 13: Oracles used by $\mathcal{A}'$ against q-SDH to answer $\mathcal{A}_x$ against signature traceability.

Reduction to the q-SDH problem. As stated before, $\mathcal{A}'$ correctly simulates all inputs to $\mathcal{A}_x$. Assume that $\mathcal{A}_x$ interacts with the oracles exposed by $\mathcal{A}'$ and, after making at most $n = q$ queries to the SNDI oracle, $\mathcal{A}_x$ outputs:

– A Σ_i tuple meeting condition 1 (i.e., $\mathcal{A}_x = \mathcal{A}_1$). Note that Σ_i cannot be associated to an honest user nor to a corrupt user via $\mathsf{Identify}(\mathsf{uid}, \Sigma_i)$, and yet, $\Sigma_i = (m_i, scp, \sigma_i, nym_i)$ is a valid signature, as Verify returns 1. Therefore, from the first equation of π_σ in σ, we know that $nym_i = \mathsf{H}(scp)^{y^*}$ for some y^*. But from the fact that $\mathsf{Identify}$ returns 0 for all $\mathsf{uid} \in \mathsf{HUL} \cup \mathsf{CUL}$, we know that $y^* \neq y_{\mathsf{uid}}$ for all y_{uid}s used to simulate the credentials for honest and corrupt users. Specifically, the signature returned by $\mathcal{A}_x$ cannot have been produced by SIGN, and we can extract a BBS+ signature from σ^*.
– A set of $(\Sigma_1, \ldots, \Sigma_n)$ meeting condition 2 (i.e., $\mathcal{A}_x = \mathcal{A}_2$). In this case, the argument is exactly as in [22, Appendix B.6]. Since the returned set is larger than the number of corrupt users, determinism of pseudonyms computation

implies that at least one $\Sigma_i = (m_i, scp, \sigma_i, nym_i)$ contains σ_i with a π_σ for $nym_i = \mathsf{H}(scp)^{y^*}$, where $y^* \neq y_{\mathsf{uid}}$ for any $\mathsf{uid} \in \mathsf{CUL}$. Also, Σ_i has not been produced by SIGN, so we can extract the witness used to produce π_σ.

The details of the extraction are as in [12], which we reproduce here for completeness. First, due to the soundness property of the proof system used to generate π_σ, $\mathcal{A}'$ extracts $(\tilde{x}, \tilde{y}, r_2, r_3, s')$ from π_σ in $\sigma^* = (A', \hat{A}, d, \pi_\sigma)$. Then, we have two cases:

- If $r_3 = 0$, from the third equation in the statement of π_σ, we know that $g_1 h_1^{\tilde{y}} h_2^{s'} = 1_{\mathbb{G}_1}$. Thus, $(\tilde{A} = 1_{\mathbb{G}_1}, \tilde{x}, s^* = s')$ is a valid BBS+ signature on $\tilde{y}$.
- If $r_3 \neq 0$, we first observe that, since the group signature verifies correctly, we know that $e(A', ipk) = e(\hat{A}, g_2)$ which, combined with the second equation in the statement of π_σ, $\hat{A} = (A')^{-\tilde{x}} h_2^{r_2} d$, gives $(A')^{\chi+\tilde{x}} = d h_2^{r_2}$. Then, from $r_3 \neq 0$ and the third equation in the statement of π_σ, $(A')^{\chi+\tilde{x}} = (g_1 h_2^{s'+r_2 r_3} h_1^{\tilde{y}})^{\frac{1}{r_3}}$. Therefore, $(\tilde{A} = (A')^{r_3}, \tilde{x}, s^* = s' + r_2 r_3)$ is a valid BBS+ signature on $\tilde{y}$.

Now, with the extracted signature $(\tilde{A}, \tilde{x}, s^*)$, there are three cases:

- If $\tilde{x} \in \{x_i\}_{0<i<q} \cup \{x_0\}$ and $\tilde{A} = A_{\mathsf{uid}}$. Since $\tilde{y} \neq y_{\mathsf{uid}}$ for all uid, then $\mathcal{A}'$ can break the discrete logarithm assumption picking any uid' and computing $\log_{h_2}(h_1) = \frac{s^* - s_{\mathsf{uid}'}}{y_{\mathsf{uid}'} - \tilde{y}}$. Indeed, when $\tilde{x} = x_i$ and $\tilde{y} \neq y_{\mathsf{uid}'}$:

$$\begin{aligned} & \tilde{A} = A_{\mathsf{uid}'} \\ \iff & g_1' h_2^{s^*} h_1^{\tilde{y}} = g_1' h_2^{s_{\mathsf{uid}'}} h_1^{y_{\mathsf{uid}'}} \\ \iff & h_2^{s^* - s_{\mathsf{uid}'}} = h_1^{y_{\mathsf{uid}'} - \tilde{y}} \\ \iff & h_2^{\frac{s^* - s_{\mathsf{uid}'}}{y_{\mathsf{uid}'} - \tilde{y}}} = h_1 \end{aligned}$$

  Since q-SDH implies the discrete logarithm assumption, $\mathcal{A}'$ can break q-SDH.
- If $\tilde{x} \in \{x_i\}_{0<i<q} \cup \{x_0\}$, but $\tilde{A} \neq A_{\mathsf{uid}}$. Assuming that $\tilde{x} = x_0$ and making $\tilde{s} \leftarrow s^* + \mu\tilde{y}$, $\mathcal{A}'$ can compute an extra SDH pair as $((\tilde{A}(g_1')^{\frac{-k\tilde{s}}{a}})^{\frac{a}{a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0)}}, \tilde{x})$:

$$\begin{aligned} & (\tilde{A}(g_1')^{\frac{-k\tilde{s}}{a}})^{\frac{a}{a-\tilde{s}}} \\ \overset{(1)}{=} & (g_1' h_2^{\tilde{s}})^{\frac{a}{(\chi+\tilde{x})(a-\tilde{s})}} (g_1')^{\frac{-k\tilde{s}}{a-\tilde{s}}} \\ \overset{(2)}{=} & (g_1' (g_1')^{\frac{(\chi+\tilde{x})k\tilde{s}-\tilde{s}}{a}})^{\frac{a}{(\chi+\tilde{x})(a-\tilde{s})}} (g_1')^{\frac{-k\tilde{s}}{a-\tilde{s}}} \\ = & (g_1')^{\frac{(\chi+\tilde{x})k\tilde{s}+a-\tilde{s}-(\chi+\tilde{x})k\tilde{s}}{(\chi+\tilde{x})(a-\tilde{s})}} \\ = & (g_1')^{\frac{1}{\chi+\tilde{x}}} \end{aligned}$$

  Where equality (1) derives from $\tilde{A} = (g_1' h_2^{\tilde{s}} h_1^{y})$, $h_1 = h_2^{\mu}$ and $\tilde{s} = s^* + \mu\tilde{y}$, and equality (2) holds since $h_2 = (d_1'(g_1')^{x_0})^k (g_1')^{-1})^{\frac{1}{a}}$.

Therefore, $\mathcal{A}'$ computes an additional q-SDH pair $((g_1')^{\frac{1}{\chi+\tilde{x}}}, \tilde{x})$, which enables it to break the q-SDH assumption as shown in [9]. However, in this case, $\tilde{x} = x_0$ only with probability $1/q$. Therefore, assuming that $\mathcal{A}_x$ succeeds with probability ϵ, $\mathcal{A}'$ breaks q-SDH with probability ϵ/q.

– If $\tilde{x} \notin \{x_i\}_{0<i<q} \cup \{x_0\}$, $\mathcal{A}'$ can compute an extra SDH pair. Making $\tilde{s} \leftarrow s^* + \mu\tilde{y}$, then $((\tilde{A}(g_1')^{\frac{-k\tilde{s}}{a}})^{\frac{a}{a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0)}}, \tilde{x})$ is an additional SDH pair, since:

$$
\begin{aligned}
& (\tilde{A}(g_1')^{\frac{-k\tilde{s}}{a}})^{\frac{a}{a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0)}} \\
\overset{(1)}{=}\ & (g_1' h_2^{\tilde{s}})^{\frac{a}{(\chi+\tilde{x})(a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0))}} (g_1')^{\frac{-k\tilde{s}}{a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0)}} \\
\overset{(2)}{=}\ & (g_1'(g_1')^{\frac{(\chi+x_0)k\tilde{s}-\tilde{s}}{a}})^{\frac{a}{(\chi+\tilde{x})(a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0))}} (g_1')^{\frac{-k\tilde{s}}{a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0)}} \\
=\ & (g_1')^{\frac{(\chi+x_0)k\tilde{s}+a-\tilde{s}-(\chi+\tilde{x})k\tilde{s}}{(\chi+\tilde{x})(a-\tilde{s}-k\tilde{s}(\tilde{x}-x_0))}} \\
=\ & (g_1')^{\frac{1}{\chi+\tilde{x}}}
\end{aligned}
$$

Where equalities (1) and (2) hold for the same reason as in the previous case. Therefore given the extra SDH pair $((g_1')^{\frac{1}{\chi+\tilde{x}}}, \tilde{x})$, $\mathcal{A}'$ can break the q-SDH assumption as shown in [9].

Overall, we have that, in the worst case, $\mathcal{A}'$ manages to break the q-SDH assumption with probability ϵ/q (either using $\mathcal{A}_1$ or $\mathcal{A}_2$).

B.4 Link Traceability

Link traceability follows from unforgeability of the BBS+ credentials and zero-knowledgeness and soundness of SPK, for winning condition 1. For winning condition 2, it follows from the discrete logarithm assumption and soundness of SPK.

Proof (Link Traceability). We describe two adversaries, $\mathcal{A}_1'$ and $\mathcal{A}_2'$, that leverage adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ breaking link traceability through winning conditions 1 or 2, respectively. $\mathcal{A}_1'$ breaks the q-SDH assumption, and $\mathcal{A}_2'$ breaks the discrete logarithm assumption.

In the following, the $\mathsf{Identify}$ function is defined exactly as in Appendix B.3.

Adversary $\mathcal{A}_1'$ against q-SDH. Since $\mathsf{VerifyLink}$ includes verification of independent signatures, this is essentially as in the first winning condition of signature traceability. Following the same strategy, $\mathcal{A}_1'$ breaks the q-SDH assumption with probability ϵ/q

Adversary $\mathcal{A}_2'$ against discrete logarithm. $\mathcal{A}_2'$ receives an instance $(\mathsf{D}_0, \mathsf{D}_1 = \mathsf{D}_0^{\mathsf{a}})$ of the discrete logarithm problem. It just programs the random oracle H as described next. The rest of the oracles are as in the link traceability game.

Random oracle H. For a newly queried input in, H uniformly chooses at random $r \leftarrow_{\$} \mathbb{Z}_p$, computes $G \leftarrow \mathsf{D}_1\mathsf{D}_0^r$, adds (in, r) to its internal list HL and

$$
\begin{array}{l}
\underline{\mathcal{A}'(\mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}})} \\
param \leftarrow \mathsf{Setup}(1^\tau) \\
(ipk, isk) \leftarrow \mathsf{IKGen}(param) \\
(lm, \boldsymbol{\Sigma}, \pi_l) \leftarrow \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SIGN},\mathsf{LINK}}(ipk, isk) \\
\textbf{if } \mathsf{VerifyLink}(ipk, lm, \boldsymbol{\Sigma}, \pi_l) = 1 \wedge \\
\quad \exists \mathsf{uid} \neq \mathsf{uid}', \Sigma \neq \Sigma' \in \boldsymbol{\Sigma} \text{ s.t.} \mathsf{Identify}(\mathsf{uid}, \Sigma) = 1 \wedge \mathsf{Identify}(\mathsf{uid}', \Sigma') = 1 : \\
\quad \mathsf{a} = \dfrac{\sum_{i \in [n]} (y - y_i) r_i}{-ny + \sum_{i \in [n]} y_i} \\
\quad \textbf{return } \mathsf{a} \\
\textbf{return } \perp
\end{array}
\qquad
\begin{array}{l}
\underline{\mathsf{H}(\mathsf{in})} \\
\textbf{if } (\mathsf{in}, r) \in \mathsf{HL} : \textbf{return } \mathsf{D}_1 \mathsf{D}_0^r \\
r \leftarrow \mathbb{Z}_p, \mathsf{HL} \leftarrow \mathsf{HL} \cup \{(\mathsf{in}, r)\} \\
\textbf{return } \mathsf{D}_1 \mathsf{D}_0^r
\end{array}
$$

Fig. 14: Adversary $\mathcal{A}_2'$ against DL from $\mathcal{A}_2$ breaking link traceability through winning condition 2.

returns G. For previously queried inputs, it responds consistently. The behaviour of H is clearly indistinguishable from random.

Reduction to Discrete Logarithm. Since H behaves randomly, and the rest of the oracles are the default ones, the simulation by $\mathcal{A}_2'$ is indistinguishable to $\mathcal{A}_2$ from real executions. Eventually, $\mathcal{A}_2$ returns a $(lm, \boldsymbol{\Sigma}, \pi_l)$ tuple. Let $\boldsymbol{\Sigma} = \{\Sigma_i = (m_i, scp_i, (A_i', \hat{A}_i, d_i, \pi_i), nym_i)\}_{i \in [n]}$. Since $\mathsf{VerifyLink}$ returns 1, we know that (1) $nym_i = \mathsf{H}(scp_i)^{y_i}$ for every Σ_i, for some y_i. As none of the π_i has been simulated, we can extract all these y_i. Again, since $\mathsf{VerifyLink}$ returns 1, we know that (2) $\prod_{i \in [n]} nym_i = \overline{nym} = \overline{hscp}^y = (\prod_{i \in [n]} \mathsf{H}(scp_i))^y$, from which we can also extract y, as π_l has not been simulated either. Additionally, fetching the corresponding queries from H, we can re-write $\mathsf{H}(scp_i)$ as $(\mathsf{D}_1 \mathsf{D}_0^{r_i}) = \mathsf{D}_0^{\mathsf{a}+r_i}$ (for unkown a), where r_i is the random value computed by H when queried with scp_i. Combining (1) and (2):

$$
\begin{aligned}
\prod_{i \in [n]} nym_i &= (\prod_{i \in [n]} \mathsf{H}(scp_i))^y \\
\iff \prod_{i \in [n]} \mathsf{H}(scp_i)^{y_i} &= \prod_{i \in [n]} \mathsf{H}(scp_i)^y \\
\iff \prod_{i \in [n]} \mathsf{D}_0^{(\mathsf{a}+r_i) y_i} &= \prod_{i \in [n]} \mathsf{D}_0^{(\mathsf{a}+r_i) y} \\
\iff \mathsf{D}_0^{\sum_{i \in [n]} (\mathsf{a}+r_i) y_i - \sum_{i \in [n]} (\mathsf{a}+r_i) y} &= 1
\end{aligned}
$$

The exponent in the left-hand side of the last equation is trivially 0 when for all i, $r_i = -\mathsf{a}$, or $y_i = y$. Since the r_i values are chosen uniformly at random by H, the probability of *all* of them being equal to $-\mathsf{a}$ is negligible (exactly, $\frac{1}{p^n}$).

Suppose that $\mathcal{A}_2$ wins the link traceability game through winning condition 2. Due to the uniqueness property of $\mathsf{Identify}$, this means that there exists $i \neq j \in [n]$ such that $y_i \neq y_j$. In that case, $\mathcal{A}_2'$ extracts a as follows:

$$
\begin{aligned}
&\mathsf{D}_0^{\sum_{i\in[n]}(\mathsf{a}+r_i)y_i-\sum_{i\in[n]}(\mathsf{a}+r_i)y} = 1\\
\iff &\sum_{i\in[n]}(\mathsf{a}+r_i)y_i - \sum_{i\in[n]}(\mathsf{a}+r_i)y = 0\\
\iff &\mathsf{a} = \frac{\sum_{i\in[n]}(y-y_i)r_i}{-ny+\sum_{i\in[n]}y_i}
\end{aligned}
$$

Where the denominator is non-zero. Otherwise, assume it is. Then, $y = \frac{1}{n}\sum_{i\in[n]} y_i$, and the second equation above becomes:

$$n\sum_{i\in[n]}(\mathsf{a}+r_i)y_i = \sum_{i\in[n]}(\mathsf{a}+r_i)(\sum_{i\in[n]} y_i)$$

But then:

- Either all r_i are equal which, by the check in the third step of $\mathsf{VerifyLink}$ and the uniqueness property of $\mathsf{Identify}$, implies that all y_i are equal. Or,
- The r_i's are not all equal, but all y_i are equal.

In both cases, contradicting the initial breaking assumption that the y_i's are not all equal. Therefore, the denominator is non-zero, and a can be extracted.

Consequently, if $\mathcal{A}_2$ wins the link traceability game through winning condition 2, with non-negligible probability ϵ, then $\mathcal{A}'_2$ can break the discrete logarithm problem with non-negligible probability $\epsilon - 1/p^n$.

B.5 Signature Non-frameability

Signature non-frameability holds after the uniqueness property of the pseudonyms and zero-knowledgness and soundness of SPK.

Proof (Signature non-frameability). We build an adversary $\mathcal{A}'$ that breaks the discrete logarithm assumption leveraging an adversary $\mathcal{A}$ against signature non-frameability.

Adversary $\mathcal{A}'$ against discrete logarithm. $\mathcal{A}'$ randomly picks the user uid^* among the up to q users $\mathcal{A}$ incorporates via SNDU, sets h_1 to D_0, and generates the remaining values in *param* as usual. Note that h_1 is uniformly random, as in the signature non-frameability game. Then, $\mathcal{A}'$ answers SNDU, SIGN and LINK queries from $\mathcal{A}$. When $\mathcal{A}$ gives a response, $\mathcal{A}'$ finds a signature by uid^* that has not been queried to SIG, extracts y^* and returns it as a response to the discrete logarithm challenge. Next, we describe why the oracles exposed by $\mathcal{A}'$ correctly simulate the oracles in the real game. Their formal definition is given in Fig. 16.

Random Oracle (H *function).* For newly queried values in, the random oracle used by $\mathcal{A}'$ computes a random number r, updates HL with the pair (in, r) and returns D_0^r as the hash value for the received input. For already queried in values, it just fetches from HL the previously computed r value for in and returns D_0^r. Note that the outputs produced in this manner are random.

$\mathcal{A}'(\mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}})$

$h_1 \leftarrow \mathsf{D}_0$, Generate remaining $param$ with $\mathsf{Setup}(1^\tau)$
$i_{\mathsf{uid}} \leftarrow 0, k_{\mathsf{uid}} \leftarrow [q]$
$(ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\Sigma = (m, scp, \sigma, nym)) \leftarrow \mathcal{A}^{\mathsf{SNDU,SIGN,LINK}}(ipk, isk)$
if $\mathsf{Verify}(ipk, \Sigma) = 1 \ \wedge \ \exists \mathsf{uid}$ s.t. $\Sigma \notin \mathsf{SIG}[\mathsf{uid}] \wedge (*, scp, *, nym) \in \mathsf{SIG}[\mathsf{uid}]$:
  if $\mathsf{uid} \neq \mathsf{uid}^*$: **return** $\perp$
  Parse Σ' as $(\cdot, \cdot, (\cdot, \cdot, \cdot, \pi^*_\sigma), \cdot)$
  Extract y^* from π^*_σ
  return y^*
return $\perp$

Fig. 15: Adversary $\mathcal{A}'$ against DL from $\mathcal{A}$ breaking signature non-frameability.

Simulating the SNDU *oracle.* $\mathcal{A}'$'s implementation of the SNDU oracle differs from the default oracle just when adding the k_{uid}-th user. For that case, the SNDU oracle initializes uid^*, sets the Y value to D_1 and simulates the proof π_Y. The zero-knowledge property of the underlying proof system ensures that the output is indistinguishable from normal executions. The *private key* of user uid^* will then be a such that $\mathsf{D}_1 = \mathsf{D}_0^{\mathsf{a}}$. Note that, even though it is unknown to $\mathcal{A}'$, it will be possible to simulate all computations in which it is involved.

Simulating the SIGN *oracle.* The SIGN oracle used by $\mathcal{A}'$ works as usual for all inputs except for queries for signatures by user uid^*. In this case, the oracle fetches from HL the random number r used to compute the hash value for scp and sets the nym to D_1^r. This is consistent with SNDU, since $Y = \mathsf{D}_0^{\mathsf{a}}$ and $nym = \mathsf{D}_1^r = \mathsf{D}_0^{\mathsf{a}r}$ for some unknown a. Also, nym is well formed, since $\mathsf{H}(scp) = \mathsf{D}_0^r$ and, due to the randomness of r, it is equally distributed to real nyms. The zero-knowledge property of the proof system ensures that π_σ can be simulated, producing a signature that is indistinguishable from real signatures.

Simulating the LINK *oracle.* The LINK oracle used by $\mathcal{A}'$ only differs from the default LINK oracle in queries where the user identifier is uid^*. In this case, $\overline{hscp}$ and $\overline{nym}$ are computed as usual, but π_l is simulated. Note that $\overline{hscp} = \prod_i \mathsf{D}_0^{r_i}$ and $\overline{nym} = \prod_i \mathsf{D}_1^{r_i} = \mathsf{D}_0^{\mathsf{a}r_i}$ and, therefore, are well formed. The zero-knowledge property of the proof system ensures that the output is indistinguishable from real outputs.

Reduction to the discrete logarithm problem. The construction by $\mathcal{A}'$ of the oracles queried by $\mathcal{A}$ ensures that all inputs that $\mathcal{A}$ gets are indistinguishable from real inputs. Assume that $\mathcal{A}$ wins the signature non-frameability game with probability ϵ and that $\mathsf{uid} = \mathsf{uid}^*$. Then, with probability ϵ, $\mathcal{A}$ outputs a valid signature $\Sigma \notin \mathsf{SIG}[\mathsf{uid}]$, but with a nym that matches the one in another signature in $\mathsf{SIG}[\mathsf{uid}]$. The scope-exclusiveness property ensured by our approach to pseudonyms, ensures that there is only one y^* such that $nym = \mathsf{H}(scp)^{y^*}$. Since the signature verifies, soundness of the SPK in the signature (which is not a simulated proof) ensures that this y^* must have been used to produce it. Therefore, $\mathcal{A}'$ can extract y^* from the proof, and return it as the solution to the discrete

Global variables (set by $\mathcal{A}'$)

$\mathsf{uid}^*, i_{\mathsf{uid}}, k_{\mathsf{uid}}$
$\mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}}$

$\mathsf{H}(\mathsf{in})$

if $(\mathsf{in}, r) \in \mathsf{HL}$: **return** D_0^r
$r \leftarrow \mathbb{Z}_p, \mathsf{HL} \leftarrow \mathsf{HL} \cup \{(\mathsf{in}, r)\}$
return D_0^r

$\mathsf{SIGN}(\mathsf{uid}, m, scp)$

Check input as default SIGN
if $\mathsf{uid} = \mathsf{uid}^*$:
  Find $(scp, r) \in \mathsf{HL}$
  $nym \leftarrow \mathsf{D}_1^r, A', d \leftarrow \mathbb{Z}_p^*, \hat{A} \leftarrow (A')^{isk}$
  Simulate π_σ with $A', \hat{A}, d, scp, nym, m$
  $\sigma \leftarrow (A', \hat{A}, d, \pi_\sigma), \Sigma \leftarrow (m, scp, \sigma, nym)$
  $\mathsf{SIG} \leftarrow \mathsf{SIG} \cup \{(\mathsf{uid}, \Sigma)\}$
  return (σ, nym)
else : Run SIGN as usual

$\mathsf{SNDU}(\mathsf{uid}, \mathsf{M}_{in})$

Check input as default SNDU
Update HUL as default SNDU
$i_{\mathsf{uid}} \leftarrow i_{\mathsf{uid}} + 1$, Parse M_{in} as n
if $i_{\mathsf{uid}} = k_{\mathsf{uid}}$:
  $\mathsf{uid}^* \leftarrow \mathsf{uid}, Y \leftarrow \mathsf{D}_1$
  Simulate π_Y with (Y, n)
  return $((Y, \pi_Y), \mathsf{cont})$
Continue from line 6 of SNDU

$\mathsf{LINK}(\mathsf{uid}, lm, \boldsymbol{\Sigma})$

Check input as default LINK
if $\mathsf{uid} = \mathsf{uid}^*$:
  $\mathsf{LNK} \leftarrow \mathsf{LNK} \cup \{(\mathsf{uid}, lm, \boldsymbol{\Sigma})\{$
  Parse $\boldsymbol{\Sigma}$ as $\{(m_i, scp_i, \sigma_i, nym_i)\}_{i\in[n]}$
  $\overline{hscp} \leftarrow \prod_{i\in[n]} \mathsf{H}(scp_i), \overline{nym} \leftarrow \prod_{i\in[n]} nym_i$
  Simulate π_l with $\overline{hscp}, \overline{nym}$
  return π_l
else Run LINK as usual

Fig. 16: Oracles used by $\mathcal{A}'$ to answer $\mathcal{A}$ against signature non-frameability.

logarithm challenge. Still, $\mathcal{A}'$ exits early when $\mathsf{uid} \neq \mathsf{uid}^*$, returning $\perp$. This happens with probability $1 - 1/q$, assuming q executions of SNDU. Therefore, the probability that $\mathcal{A}'$ computes the discrete logarithm from $\mathcal{A}$ winning the non-frameability game with probability ϵ is given by: $\mathbf{Adv}_{\mathcal{A}'}^{DL}(\tau) = \epsilon/q$, which is non-negligible if ϵ is non-negiligible.

B.6 Link Non-frameability

Link non-frameability follows from the uniqueness property of pseudonyms, and zero-knowledgeness and soundness of SPK, under the discrete logarithm assumption.

Proof (Link Non-frameability). We describe the two adversaries $\mathcal{A}_1'$ and $\mathcal{A}_2'$ that leverage adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ breaking link non-frameability using condition 1 or 2, respectively, to break the discrete logarithm problem.

Adversary $\mathcal{A}_1'$ against discrete logarithm. $\mathcal{A}_1'$ receives an instance $(\mathsf{D}_0, \mathsf{D}_1 = \mathsf{D}_0^{\mathsf{a}})$ of the discrete logarithm problem. It programs the random oracle H as described next. The rest of the oracles are as in the link non-frameability game.

Random oracle H. For a newly queried input in, H uniformly chooses at random $r \leftarrow_\$ \mathbb{Z}_p$, computes $G \leftarrow \mathsf{D}_1\mathsf{D}_0^r$, adds (in, r) to its internal list HL and returns G. For previously queried inputs, it responds consistently. The behaviour of H is clearly indistinguishable from random.

Reduction to Discrete Logarithm. Since H behaves randomly, and the rest of the oracles are the default ones, the simulation by $\mathcal{A}_1'$ is indistinguishable to $\mathcal{A}_1$ from real executions. Assume that $\mathcal{A}_1$ wins the link non-frameability game through winning condition 1 with probability ϵ. Note that the link proof π_l and all signatures in $\boldsymbol{\Sigma}$ have either been produced by the real oracle (specifically, not simulated) or by the adversary. Therefore, after soundness of SPK, for every $(m_i, scp_i, \sigma_i, nym_i) = \Sigma_i \in \boldsymbol{\Sigma}$, $nym_i = \mathsf{H}(scp_i)^{y_i}$, for some y_i which we can extract. Furthermore, given the construction of H, $\mathsf{H}(scp_i) = \mathsf{D}_0^{\mathsf{a}+r_i}$ and $nym_i = \mathsf{D}_0^{(\mathsf{a}+r_i)y_i}$, for some r_i picked by H. Since π_l verifies correctly, we also know that $\mathsf{D}_0^{y\sum_{i\in[n]}(\mathsf{a}+r_i)} = \overline{hscp}^y = \overline{nym} = \mathsf{D}_0^{\sum_{i\in[n]}(\mathsf{a}+r_i)y_i}$, for some y that can be extracted – again, because SPK is sound. This is exactly the same as in the second winning condition of the link traceability game. Therefore, arguing as we did there, $\mathcal{A}_1'$ can compute the solution to the discrete logarithm problem as $\mathsf{a} = \frac{\sum_{i\in[n]}(y-y_i)r_i}{-ny+\sum_{i\in[n]} y_i}$.

Adversary $\mathcal{A}_2'$ against discrete logarithm. $\mathcal{A}_2'$ prepares the environment and configures the oracles as the adversary against signature non-frameability.

Reduction to the discrete logarithm problem. As described in the case of signature non-frameability, the outputs by the simulated oracles are indistinguishable from the real oracles. Assume that $\mathcal{A}_2$ wins the link non-frameability game, through winning condition 2, with probability ϵ. Since the produced link proof π_l is valid and has not been simulated, we can extract y from it such that $\overline{hscp}^y = \overline{nym}$. In addition, since all the signatures in $\boldsymbol{\Sigma} = \{\Sigma_i\}_{i\in[n]}$ have been produced by the simulated SIGN oracle, we also know that $\overline{hscp} = \mathsf{D}_0^{\sum_{i\in[n]} r_i}$, and $\overline{nym} = \mathsf{D}_1^{\sum_{i\in[n]} r_i} = \mathsf{D}_0^{\mathsf{a}\sum_{i\in[n]} r_i}$, for some r_i produced by H. Therefore, $\overline{hscp}^y = \mathsf{D}_0^{y\sum_{i\in[n]} r_i} = \mathsf{D}_0^{\mathsf{a}\sum_{i\in[n]} r_i}$, and $y = \mathsf{a}$. Asuming q executions of the SNDU oracle, the probability that $\mathcal{A}_2$ picks uid^* is $1/q$. Then, the probability of $\mathcal{A}_2'$ breaking the discrete logarithm is given by $\mathbf{Adv}^{DL}_{\mathcal{A}_2'}(\tau) = \epsilon/q$, which is non-negligible if ϵ is non-negligible.

C Proofs of Security for $\boldsymbol{\Pi}_{\mathsf{sUCL}}$

C.1 Correctness

Signature correctness follows from the equivalent in the basic setting as, for verification of individual signatures, the sequence information is ignored. Next, we prove correctness of sequential links.

Correctness of Sequential Link. This property is formalized in Fig. 17. The adversary wins if it returns a set of $\{(m_i, scp_i)\}_{i\in[n]}$ pairs for which a matching set of honestly generated signatures and an honestly generated proof of sequential linking π_{seq} do not verify.

Experiment: $\mathbf{Exp}^{\mathsf{corr\text{-}slink}}_{\mathcal{A},\mathsf{sUCL}}(\tau)$

$param \leftarrow \mathsf{Setup}(1^\tau)$
$(ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}, \{m_i, scp_i\}_{i\in[n]}, lm) \leftarrow \mathcal{A}^{\mathsf{ADDU}}(ipk)$
for $i \in [n] : (\tilde{\sigma}_i, nym_i, i+1) \leftarrow \mathsf{SSign}(ipk, \mathsf{USK}[\mathsf{uid}], i, m_i, scp_i)$
$\pi_{seq} \leftarrow \mathsf{SLink}(ipk, \mathsf{USK}[\mathsf{uid}], 1, lm, \{(m_i, scp_i, \tilde{\sigma}_i, nym_i)\}_{i\in[n]})$
if $\mathsf{VerifySLink}(ipk, lm, \{(m_i, scp_i, \tilde{\sigma}_i, nym_i)\}_{i\in[n]}, \pi_{seq}) = 0$:
  return 1
return 0

Fig. 17: Correctness of sequential link experiment.

Proof (Correctness of Sequential Link). Let (A, x, y, s, k) be the secret key of user uid, and $\{(m_i, scp_i, (\sigma_i, (seq_{i,1}, seq_{i,2}, seq_{i,3})), nym_i)\}_{i\in[n]}$ and $\pi_{seq} = \{x_i\}_{i\in[n]}$ respectively be a set of signed messages and proof of them being sequentially linked, all created honestly by uid. We focus on the seq component, correctness of all the other parts following directly as in $\mathbf{Exp}^{\mathsf{corr\text{-}link}}_{\mathcal{A},\mathsf{UCL}}$. Since every $(seq_{i,1}, seq_{i,2}, seq_{i,3},)$ is generated honestly, we know that, for every i, $n_i = \mathsf{PRF.Eval}(k, 0||i)$ and $x_i = \mathsf{PRF.Eval}(k, 1||i)$, and that $seq_{i,1} = \mathsf{H}'(x_i)$, $seq_{i,2} = \mathsf{H}'(x_i \oplus x_{i-1})$ and, also $seq_{i,3} = n_i$. Therefore, $\pi_{seq} = (\pi_l, \{x_i\}_{i\in[n]})$ is correctly verified by $\mathsf{VerifySLink}$.

C.2 Anonymity

Anonymity in Π_{sUCL} follows from zero-knowledgeness and simulation-soundness of SPK, indistinguishability of the pseudonyms from random numbers, and the pseudorandomness property of PRF, under the DDH assumption and in the random oracle model.

Proof (Anonymity). We prove anonymity in Π_{sUCL} by building on the anonymity reduction used in the Π_{UCL} through a hybrid approach. Basically, we define 3 games: Game 1 is the $\mathbf{Exp}^{\mathsf{sanon}\text{-}b}_{\mathcal{A},\mathsf{sUCL}}$ definition; Game 2 is as the anonymity reduction for Π_{UCL}, but adding the new seq elements to the signatures; finally, Game 3 replaces the seq elements in signatures by the challenge user with random numbers. We define them next.

Game 1. This is $\mathbf{Exp}^{\mathsf{sanon}\text{-}b}_{\mathcal{A},\mathsf{sUCL}}$.

Game 2. This is the anonymity reduction for Π_{UCL} described in Fig. 9 of Appendix B.2, except that the oracles are extended to include the new elements. Specifically:

- The SSIGN oracle, when called for uid^*, uses the PRF to compute the x_i and x_{i-1} values that are then hashed to produce $seq = (seq_1, seq_2, seq_3)$ as in real calls to Sign. seq is added to the produced signature.
- The $\mathsf{CH\text{-}SSIGN}_b$ oracle is extended similarly as SSIGN.

– The SLINK and $\mathsf{CH\text{-}SLINK}_b$ oracles now also return the x_i values corresponding to the signatures being linked.

Indistinguishability between Game 1 and Game 2 is, as in the anonymity reduction in Appendix B.2, ensured by the DDH assumption.

Game 3. This is as Game 2, but the x_i values produced by SSIGN for uid^*, and by $\mathsf{CH\text{-}SSIGN}_b$, are replaced by random numbers.

Indistinguishability between Game 2 and Game 3 is ensured by the pseudorandom property of the PRF.

Consequently, given an adversary that is able to distinguish, with probability ϵ_{PRF}, pseudorandom numbers generated with PRF from truly random numbers, the total advantage of an adversary against anonymity in Π_{sUCL} is $\epsilon_{DDH} + \epsilon_{\mathsf{PRF}}$, which is negligible under the DDH assumption and pseudorandomness of PRF.

C.3 Sequentiality

Sequentiality in our Π_{sUCL} follows from collision resistance and preimage resistance of H', and pseudorandomness of PRF.

Proof (Sequential Link Order). We directly describe how $\mathcal{A}'$ uses the output produced by $\mathcal{A}$ to find a collision depending on the winning condition. The oracles exposed by $\mathcal{A}$ are all the default oracles.

Reduction to Collision Resistance. Assume $\mathcal{A}$ produces a $(lm^*, \boldsymbol{\Sigma}^*, \pi^*_{seq})$ tuple winning the $\mathsf{sequential}$ game. Then, it has found a sequence of signatures within $\overline{\Sigma} = \boldsymbol{\Sigma}' \cup \mathsf{SIG}[\mathsf{uid}^*]$ that is incorrectly ordered with respect to $\mathsf{SIG}[\mathsf{uid}^*]$, but for which the produced proof is valid. We describe how to extract collisions for H' in three basic cases, and then argue that (at least) one of these three cases must arise if $\mathcal{A}$ wins the game. The corresponding adversary $\mathcal{A}'$ finding collisions from $\mathcal{A}$ breaking the sequential link order property is given in Fig. 18.

Herafter, we use Σ_i to denote honestly generated signatures appearing in $\mathsf{SIG}[\mathsf{uid}^*]$, which can be parsed as $(\cdot, \cdot, (\cdot, (seq_{i,1}, seq_{i,2}, seq_{i,3}), \cdot)$; Σ^*_i to denote signatures in $\boldsymbol{\Sigma}^*$, which can be parsed as $(\cdot, \cdot, (\cdot, (seq^*_{i,1}, seq^*_{i,2}, seq^*_{i,3}), \cdot)$; and Σ'_i to denote signatures in $\boldsymbol{\Sigma}'$, which can be parsed as $(\cdot, \cdot, (\cdot, (seq'_{i,1}, seq'_{i,2}, seq'_{i,3}), \cdot)$. The x values used to compute the seq_i components of honest signatures are denoted with x_i, and the x values revealed by $\mathcal{A}$ in π^*_{seq} are denoted with x^*_i. The sizes of each set will be denoted with n for $\mathsf{SIG}[\mathsf{uid}^*]$, n^* for $\boldsymbol{\Sigma}^*$, and n' for $\boldsymbol{\Sigma}'$.

– *Skipped honest signatures.* $\boldsymbol{\Sigma}^*$ contains $(\Sigma^*_i, \Sigma^*_{i+1})$, in that order, such that in $\mathsf{SIG}[\mathsf{uid}^*]$, $\Sigma_j = \Sigma^*_i$ and $\Sigma_{j+k} = \Sigma^*_{i+1}$, for $k > 1$ (i.e., Σ_{j+k} is the k-th signature in $\mathsf{SIG}[\mathsf{uid}^*]$ after Σ_j.)
  Note that, since π^*_{seq} is a valid proof, $seq^*_{i+1,1} = \mathsf{H}'(x^*_{i+1})$, $seq^*_{i+1,2} = \mathsf{H}'(x^*_{i+1} \oplus x^*_i)$, and $x^*_{i+1} \neq x^*_i$, because the corresponding signatures are accepted by the Bulletin Board. Also, since Σ_{j+k} was honestly generated, $seq_{j+k,1} =$

$\mathsf{H}'(x_{j+k})$ and $seq_{j+k,2} = \mathsf{H}'(x_{j+k} \oplus x_{j+k-1})$, where $\mathcal{A}'$ can (re)compute x_{j+k} and x_{j+k-1}. Then, if $x_{j+k} \neq x^*_{i+1}$, (x_{j+k}, x^*_{i+1}) is a collision for $seq_{j+k,1} = seq^*_{i+1,1}$. Otherwise, if $x_{j+k} = x^*_{i+1}$, $\mathcal{A}'$ compares x_j and x^*_i. If $x_j \neq x^*_i$, then (x_j, x^*_i) is a collision for $seq_{j,1} = seq^*_{i,1}$. But, if $x_j = x^*_i$, then $x_{j+k-1} \neq x_j = x^*_i$, because $k > 1$ and therefore $j + k - 1 > j$, so $x_{j+k} \oplus x_{j+k-1} \neq x^*_{i+1} \oplus x^*_i$ and $(x_{j+k} \oplus x_{j+k-1}, x^*_{i+1} \oplus x^*_i)$ is a collision for $seq_{j+k,2} = seq^*_{i+1,2}$.

- *Swapped honest signatures.* $\boldsymbol{\Sigma}^*$ contains $(\Sigma^*_i, \Sigma^*_{i+1})$, in that order, such that $\Sigma^*_i = \Sigma_{j+k}$ and $\Sigma^*_{i+1} = \Sigma_j$, for $k \geq 1$, in $\mathsf{SIG}[\mathsf{uid}^*]$ (i.e., Σ_{j+k} is the k-th signature in $\mathsf{SIG}[\mathsf{uid}^*]$ after Σ_j.)

  Again, since π^*_{seq} is a valid proof, and the signature is accepted by the Bulletin Board, $seq^*_{i+1,1} = \mathsf{H}'(x^*_{i+1})$, $seq^*_{i+1,2} = \mathsf{H}'(x^*_{i+1} \oplus x^*_i)$, and $x^*_{i+1} \neq x^*_i$. Also, since Σ_{j+k} was honestly generated, $seq_{j+k,1} = \mathsf{H}'(x_{j+k})$ and $seq_{j+k,2} = \mathsf{H}'(x_{j+k} \oplus x_{j+k-1})$, where $\mathcal{A}'$ can recompute x_{j+k}, x_{j+k-1}. If $x_j \neq x^*_{i+1}$, then (x_j, x^*_{i+1}) is a collision for H', since $seq_{j,1} = \mathsf{H}(x_j) = \mathsf{H}(x^*_{i+1}) = seq^*_{i+1,1}$. If $x_j = x^*_{i+1}$, $\mathcal{A}'$ compares x_{j+k} and x^*_i. Suppose $x_{j+k} \neq x^*_i$. Then, (x_{j+k}, x^*_i) is a collision for H', since $seq_{j+k,1} = \mathsf{H}'(x_{j+k}) = \mathsf{H}'(x^*_i) = seq^*_{i,1}$. Otherwise, if $x_{j+k} = x^*_i$, then $x_{j-1} \neq x^*_i$, because x_{j-1} and x_{j+k} are both honestly generated and therefore $x_{j-1} \neq x_{j+k} = x^*_i$, since $j + k > j - 1$. Therefore, $x_j \oplus x_{j-1} \neq x^*_{i+1} \oplus x^*_i = x_j \oplus x_{j+k}$, and $(x_j \oplus x_{j-1}, x^*_{i+1} \oplus x^*_i)$ is a collision for H', since $seq_{j,2} = \mathsf{H}'(x_j \oplus x_{j-1}) = \mathsf{H}'(x^*_{i+1} \oplus x^*_i) = seq^*_{i+1,2}$.

- *Inserted signatures (from $\boldsymbol{\Sigma}'$.)* $\boldsymbol{\Sigma}^*$, of size n^*, contains either (a) $(\Sigma^*_1, \Sigma^*_2, \dots)$, (b) $(\dots, \Sigma^*_{i-1}, \Sigma^*_i, \Sigma^*_{i+1} \dots)$, or (c) $(\dots, \Sigma^*_{n^*-1}, \Sigma^*_{n^*})$ where, respectively, $\Sigma^*_1 = \Sigma'_j$, $\Sigma^*_i = \Sigma'_j$ and $\Sigma^*_{n^*} = \Sigma'_j$, for some $i \in [2, n^* - 1]$ and $j \in [n']$.

  For (a), we asume without loss of generality that $\Sigma^*_2 \in \mathsf{SIG}[\mathsf{uid}^*]$. Otherwise, we repeat the following argument, but setting Σ^*_1 to Σ^*_2 and Σ^*_2 to Σ^*_3, until $\Sigma^*_2 \in \mathsf{SIG}[\mathsf{uid}^*]$, which must happen, as $\boldsymbol{\Sigma}^* \cap \mathsf{SIG}[\mathsf{uid}^*] \neq \emptyset$. Since π^*_{seq} is a valid proof, $seq^*_{2,1} = \mathsf{H}'(x^*_2)$ and $seq^*_{2,2} = \mathsf{H}'(x^*_2 \oplus x^*_1)$, where $x^*_2 \neq x^*_1$. If $\Sigma^*_2 = \Sigma_\ell$, then if $x_\ell \neq x^*_2$, we have a collision in $seq_{\ell,1} = \mathsf{H}'(x_\ell) = \mathsf{H}'(x^*_2) = seq^*_{2,1}$. Otherwise, when $x_\ell = x^*_2$ observe that, when $x_{\ell-1}$ is computed to produce $\Sigma_{\ell-1}$, uid^* has not yet been corrupted and is therefore obtained through the PRF. Also, when $seq'_j = seq^*_1$ is computed by $\mathcal{A}$, uid^* has not been corrupted and $\mathcal{A}$ has no knowledge of $x_{\ell-1}$ due to the pseudorandomness property of PRF (and the preimage resistance property of H', if $\mathcal{A}$ produces seq'_j after $seq_{\ell-1}$.) Additionally, after the preimage resistance property of H', we can assume that $\mathcal{A}$ computes $seq'_{j,1}$ by first arbitrarily computing x'_j and making $seq'_{j,1} \leftarrow \mathsf{H}'(x'_j)$ as, otherwise, $\mathcal{A}$ will not be able to produce a preimage for π^*_{seq}. Therefore, with overwhelming probability, $x^*_1 = x'_j$ and, under the pseudorandomness property of the PRF in our construction, the probability that $x^*_1 = x_{\ell-1}$ when $\mathcal{A}$ computes it is negligible, making the probability that $\mathcal{A}$ finds a matching sequence also negligible.

  Assuming that at least one signature in $\mathsf{SIG}[\mathsf{uid}^*]$ follows $\Sigma^*_i = \Sigma'_j$, the analysis for case (b) is analogous to that of case (a), but ignoring the signatures preceeding Σ^*_i in $\boldsymbol{\Sigma}^*$. Otherwise, assuming that at least one signature in $\mathsf{SIG}[\mathsf{uid}^*]$ preceeds $\Sigma^*_i = \Sigma'_j$, we can proceed as in case (c), analysed next. Note that, since $\mathsf{SIG}[\mathsf{uid}^*] \cap \boldsymbol{\Sigma}^* \neq \emptyset$, at least one of them must occur.

For case (c), since π^*_{seq} is a valid proof, then $seq^*_{n^*,1} = \mathsf{H}'(x^*_{n^*})$ and $seq^*_{n^*,2} = \mathsf{H}'(x^*_{n^*} \oplus x^*_{n^*-1})$, for $x^*_{n^*} \neq x^*_{n^*-1}$. Following a similar argument as in (a), we assume without loss of generality that $\Sigma^*_{n^*-1} = \Sigma_\ell \in \mathsf{SIG}[\mathsf{uid}^*]$. Otherwise, ignore $\Sigma^*_{n^*}$ and take $\Sigma^*_{n^*-1}$ to be "the new" $\Sigma^*_{n^*}$, until $\Sigma^*_{n^*-1} \in \mathsf{SIG}[\mathsf{uid}^*]$, which must eventually occur. Since $\Sigma_\ell \in \mathsf{SIG}[\mathsf{uid}^*]$, then $seq_{\ell,1} = \mathsf{H}'(x_\ell)$, for some x_ℓ value that $\mathcal{A}'$ can recompute as needed. Suppose then that the $x^*_{n^*-1}$ value used by $\mathcal{A}$ in π^*_{seq} is different than x_ℓ (which $\mathcal{A}$ can only reproduce *after* corrupting uid^*). Then, $(x_\ell, x^*_{n^*-1})$ is a collision for $seq_{\ell,1} = \mathsf{H}'(x_\ell) = \mathsf{H}'(x^*_{n^*-1}) = seq^*_{n^*-1,1}$. Otherwise, assume that $x^*_{n^*-1} = x_\ell$. Then, under the preimage resistance property of H', $x^*_{n^*}$ must actually be the value used to compute $seq^*_{n^*,1} = seq'_{j,1}$, i.e., $seq'_{j,1} = \mathsf{H}'(x^*_{n^*})$, and $seq'_{j,2} = \mathsf{H}'(x^*_{n^*} \oplus x^*_{n^*-1}) = \mathsf{H}'(x^*_{n^*} \oplus x_\ell)$. However, note that, when $\mathcal{A}$ computes seq'_j, under the pseudorandomness property of the used PRF (and preimage resistance of H', if seq'_j is computed after Σ_ℓ), $\mathcal{A}$ has no knowledge about x_j, except with negligible probability, and therefore this only happens with negligible probability.

To complete the argument recall first that, in our construction, all signatures are uploaded to an append-only bulletin board. At this point, signatures are independently verified and, moreover, the bulletin board checks that no *seq* info is repeated across signatures. Then, observe that $\mathcal{A}$ is restricted to produce $\boldsymbol{\Sigma}^*$ using signatures from $\boldsymbol{\Sigma}'$ and $\mathsf{SIG}[\mathsf{uid}^*]$. Therefore, all $\mathcal{A}$ can do is remove signatures from $\mathsf{SIG}[\mathsf{uid}^*]$, and then, we can apply the analysis in the *Skipped honest signatures* case; permute signatures in $\mathsf{SIG}[\mathsf{uid}^*]$, and we can apply the analysis in *Swapped honest signatures* or *Skipped honest signatures*; or, finally, insert in any arbitrary location and as many times as it wants, dishonest signatures from $\boldsymbol{\Sigma}'$, and we can apply the argument in *Inserted signatures from* $\boldsymbol{\Sigma}'$, as at least one signature from $\mathsf{SIG}[\mathsf{uid}^*]$ must be present in $\boldsymbol{\Sigma}^*$. Consequently, as shown in the analysis of the individual cases, except for cases that only occur with negligible probability, $\mathcal{A}'$ finds a collision for H'.

D Generalized Expanded DDH Self-Reduction

Next, we generalize in a trivial manner the Expanded DDH Self-Reduction given in [29], providing an algorithm that allows us to obtain any arbitrary number of DDH challenges from a single DDH challenge. This is used in the reduction of the anonymity property in our scheme. The algorithm, $\mathsf{DDHRerand}_t$, is given in Fig. 9.

We extend the proof in [29] to our generic case. Note the variable renaming with respect to [29] (to ensure consistency with the naming conventions used in this paper.)

Theorem 3 (Generalized Expanded DDH Self-Reduction [29]). *Let* $(p, \mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}}, \mathsf{D}_2 = g^{\mathsf{b}}, \mathsf{D}_3 = g^{\mathsf{c}})$ *define a DDH problem instance. If* $(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ *is a DDH tuple,* $\mathsf{DDHRerand}_t$ *produces* t *DDH tuples of the form* $(\tilde{\mathsf{D}}_0 = g, \tilde{\mathsf{D}}_1 =$

$\mathcal{A}'(\mathsf{H}')$

$param \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(param)$
$(\mathsf{uid}^*, \boldsymbol{\Sigma}', \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{ADDU,SNDI,SSIGN,SLINK}}(choose, ipk)$
if $\mathsf{uid}^* \notin \mathsf{HUL} \vee \mathsf{USK}[\mathsf{uid}^*] = \perp$: **return** 0
$\mathsf{HUL} \leftarrow \mathsf{HUL} \setminus \{\mathsf{uid}^*\}, \mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{\mathsf{uid}^*\}$
$(lm^*, \boldsymbol{\Sigma}^*, \pi^*_{seq}) \leftarrow \mathcal{A}^{\mathsf{ADDU,SNDI,SSIGN,SLINK}}(forge, \mathsf{state}, \mathsf{USK}[\mathsf{uid}^*], st_{\mathsf{uid}^*})$
$\overline{\boldsymbol{\Sigma}} \leftarrow \boldsymbol{\Sigma}' \cup \mathsf{SIG}[\mathsf{uid}^*]$
// *Skipped honest signatures*
if $\exists \Sigma_j, \Sigma_{j+k} \in \mathsf{SIG}[\mathsf{uid}^*], k > 1, (\Sigma^*_i, \Sigma^*_{i+1}) \in \boldsymbol{\Sigma}^*$ s.t. $\Sigma^*_i = \Sigma_j \wedge \Sigma^*_{i+1} = \Sigma_{j+k}$:
  if $x_{j+k} \neq x^*_{i+1}$: **return** (x_{j+k}, x^*_{i+1})
  else :
    if $x_j \neq x^*_i$: **return** (x_j, x^*_i)
    else : **return** $(x_{j+k} \oplus x_{j+k-1}, x^*_{i+1} \oplus x^*_i)$
// *Swapped honest signatures*
if $\exists \Sigma_j, \Sigma_{j+k} \in \mathsf{SIG}[\mathsf{uid}^*], k \geq 1, (\Sigma^*_i, \Sigma^*_{i+1}) \in \boldsymbol{\Sigma}^*$ s.t. $\Sigma^*_i = \Sigma_{j+k} \wedge \Sigma^*_{i+1} = \Sigma_j$:
  if $x_j \neq x^*_{i+1}$: **return** (x_j, x^*_{i+1})
  else :
    if $x_{j+k} \neq x^*_i$: **return** (x_{j+k}, x^*_i)
    else : **return** $(x_j \oplus x_{j-1}, x^*_{i+1} \oplus x^*_i)$
// *Inserted signatures (case (a))*
if $\exists \Sigma_\ell \in \mathsf{SIG}[\mathsf{uid}^*], \Sigma'_j \in \boldsymbol{\Sigma}', (\Sigma^*_1, \Sigma^*_2) \in \boldsymbol{\Sigma}^*$ s.t. $\Sigma^*_1 = \Sigma'_j \wedge \Sigma^*_2 = \Sigma_\ell \in \mathsf{SIG}[\mathsf{uid}^*]$:
  if $x_\ell \neq x^*_2$: **return** (x_ℓ, x^*_2)
// *Inserted signatures (case (c))*
if $\exists \Sigma_\ell \in \mathsf{SIG}[\mathsf{uid}^*], \Sigma'_j \in \boldsymbol{\Sigma}', (\Sigma^*_{n^*-1}, \Sigma^*_{n^*}) \in \boldsymbol{\Sigma}^*$ s.t. $\Sigma^*_{n^*-1} = \Sigma_\ell \wedge \Sigma^*_{n^*} = \Sigma'_j$:
  if $x_\ell \neq x^*_{n^*-1}$: **return** $(x_\ell, x^*_{n^*-1})$
return $\perp$

Fig. 18: Adversary $\mathcal{A}'$ against collision resistance of H' from $\mathcal{A}$ breaking the sequential link order property.

$g^{\tilde{\mathsf{a}}_i}, \tilde{\mathsf{D}}_2 = g^{\tilde{\mathsf{b}}}, \tilde{\mathsf{D}}_3 = g^{\tilde{\mathsf{c}}_i}$); *or a random* $(2t+2)$*-tuple if* $(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ *is not a DDH tuple.*

Proof. (Generalized Expanded DDH Self-Reduction [29]). We distinguish two cases, depending on whether the input is a DDH tuple or not.

Case 1. Suppose $(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ is a DDH tuple, i.e., $(\mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}}, \mathsf{D}_2 = g^{\mathsf{b}}, \mathsf{D}_3 = g^{\mathsf{ab}})$. Then, for every $i \in [1, t]$, $\tilde{\mathsf{D}}_{t+i+1} = g^{u_i \mathsf{ab} + w_i \mathsf{b} + u_i v \mathsf{a} + w_i v}$. Letting $\tilde{\mathsf{a}}_i = u_i \mathsf{a} + w_i$ and $\tilde{\mathsf{b}} = \mathsf{b} + v$, we have that $\tilde{\mathsf{a}}_i \mathsf{b} = u_i \mathsf{ab} + u_i \mathsf{a} v + w_i \mathsf{b} + w_i v$. Therefore $\tilde{\mathsf{D}}_{t+i+1} = g^{\tilde{\mathsf{a}}_i \tilde{\mathsf{b}}}$ and $(\tilde{\mathsf{D}}_0 = g, \tilde{\mathsf{D}}_i = g^{\tilde{\mathsf{a}}_i}, \tilde{\mathsf{D}}_{t+1} = g^{\tilde{\mathsf{b}}}, \tilde{\mathsf{D}}_{t+i+1} = g^{\tilde{\mathsf{a}}_i \tilde{\mathsf{b}}})$ is a DDH tuple.

Case 2. Suppose $(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ is not a DDH tuple, i.e., $(\mathsf{D}_0 = g, \mathsf{D}_1 = g^{\mathsf{a}}, \mathsf{D}_2 = g^{\mathsf{b}}, \mathsf{D}_3 = g^{\mathsf{c}} = g^{\mathsf{ab}+r})$ for some $r \neq 0$. We have that, for $i \in [t]$, $\tilde{\mathsf{D}}_{t+i+1} = g^{u_i \mathsf{c} + w_i \mathsf{b} + u_i v \mathsf{a} + w_i v} = g^{u_i \mathsf{ab} + u_i r + w_i \mathsf{b} + u_i v \mathsf{a} + w_i v} = g^{(u_i \mathsf{a} + w_i)(\mathsf{b}+v) + u_i r}$. Letting $\tilde{\mathsf{a}}_i = u_i \mathsf{a} + w_i$ and $\tilde{\mathsf{b}} = \mathsf{b} + v$, $g^{(u_i \mathsf{a} + w_i)(\mathsf{b}+v) + u_i r} = g^{\tilde{\mathsf{a}}_i \tilde{\mathsf{b}} + u_i r}$. Where, since

$u_i, r \neq 0$, $g^{\tilde{\mathsf{a}}_i\tilde{\mathsf{b}}+u_i r} \neq g^{\tilde{\mathsf{a}}_i\tilde{\mathsf{b}}}$ is a random element. Consequently, $(\tilde{\mathsf{D}}_0 = g, \tilde{\mathsf{D}}_i = g^{u_i\mathsf{a}+w_i}, \tilde{\mathsf{D}}_{t+1} = g^{\mathsf{b}+v}, \tilde{\mathsf{D}}_{t+i+1} = g^{\tilde{\mathsf{a}}_i\tilde{\mathsf{b}}+u_i r} \neq g^{\tilde{\mathsf{a}}_i\tilde{\mathsf{b}}})$ is not a DDH tuple.

Since all u_i, w_i values are chosen uniformly at random within $\mathbb{Z}_p^*$, and only used once for the corresponding tuple, all tuples will be different with overwhelming probability. In consequence, when $(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ is a DDH tuple, for $i \in [t]$, every tuple composed by $(\tilde{\mathsf{D}}_0, \tilde{\mathsf{D}}_i, \tilde{\mathsf{D}}_{t+1}, \tilde{\mathsf{D}}_{t+i+1})$ will also be a (different) DDH tuple. Conversely, when $(\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ is not a DDH tuple, then every $(\tilde{\mathsf{D}}_0, \tilde{\mathsf{D}}_i, \tilde{\mathsf{D}}_{t+1}, \tilde{\mathsf{D}}_{t+i+1})$ will be a (different) not DDH tuple; or, equivalently $(\tilde{\mathsf{D}}_0, ..., \tilde{\mathsf{D}}_{2t+1})$ is a random $(2t+2)$-tuple.