Group Signatures with Selective Linkability

Lydia Garms[1]* and Anja Lehmann[2]

[1] Royal Holloway, University of London, UK
Lydia.Garms.2015@live.rhul.ac.uk
[2] IBM Research – Zurich, Switzerland
anj@zurich.ibm.com

Abstract. Group signatures allow members of a group to anonymously produce signatures on behalf of the group. They are an important building block for privacy-enhancing applications, e.g., enabling user data to be collected in authenticated form while preserving the user's privacy. The linkability between the signatures thereby plays a crucial role for balancing utility and privacy: knowing the correlation of events significantly increases the utility of the data but also severely harms the user's privacy. Therefore group signatures are unlinkable per default, but either support linking or identity escrow through a dedicated central party or offer user-controlled linkability. However, both approaches have significant limitations. The former relies on a fully trusted entity and reveals too much information, and the latter requires exact knowledge of the needed linkability at the moment when the signatures are created. However, often the exact purpose of the data might not be clear at the point of data collection. In fact, data collectors tend to gather large amounts of data at first, but will need linkability only for selected, small subsets of the data. We introduce a new type of group signature that provides a more flexible and privacy-friendly access to such selective linkability. When created, all signatures are fully unlinkable. Only when strictly needed or desired, should the required pieces be made linkable with the help of a central entity. For privacy, this linkability is established in an oblivious and non-transitive manner. We formally define the requirements for this new type of group signatures and provide an efficient instantiation that provably satisfies these requirements under discrete-logarithm based assumptions.

1 Introduction

Group signatures are a powerful and well-studied primitive that allow members of a group to sign messages on behalf of the group in an anonymous way [22, 4, 9, 21, 2, 32, 28, 34, 10, 33]. That is, a verifier of a group signature is assured that it was signed by a valid member of the group, but it does not learn anything about the identity of the signer, or even whether two signatures stem from the same user. This makes group signatures highly suited whenever data is collected that needs to be authenticated while, at the same time, the privacy of the data sources must be respected and preserved. In particular when data is

* Work done as an intern at IBM Research – Zurich.

collected from users, the protection of their privacy is of crucial importance, and sees increased attention due to the recently introduced General Data Protection Regulation (GDPR) [1], Europe's new privacy regulation. In fact, the GDPR creates strong incentives for data collectors to thoroughly protect users' data and implement the principle of data minimization, as data breaches are fined with up to 4% of an enterprises annual turnover.

When aiming to implement such techniques for privacy and data protection, one needs to find a good balance with utility though: data gets collected in order to be analysed and to generate new insights. For these processes it is usually necessary to know the correlation among different data events, as they carry a crucial part of the information. For instance, when a group of users measure and upload their blood pressure via wearable activity trackers, several high value measurements are not critical when they are distributed over many participants, but might be alarming when originating from a single user.

Often the exact purpose of the data might not be clear at the point of data collection. In fact, given the rapid advancements in machine learning and the ubiquitously available and cheap storage, data collectors tend to gather large amounts of data at first, and will only use small subsets for particular applications as they arise. A famous example are the Google Street View cars that inadvertently recorded public Wi-Fi data like SSID information, which later got used to improve Google's location services.

Ideally, the data should be collected and stored in authenticated and unlinkable form, and only the particular subsets that are needed later on should be correlated in a controlled and flexible manner.

Linkability in Group Signatures. To address the tension between privacy and utility, group signatures often have built-in measures that control linkability of otherwise anonymously authenticated information. Interestingly, despite the long line of work on this subject, none of the solutions provides the functionality to cater for the flexibility needed in practice: They either recover linkablity in a privacy-invasive way or offer control only in a static manner.

Group Signatures with Opening. Standard group signatures [22, 9, 4, 5, 10, 34] guarantee full unlinkability of signatures, except to the group manager (or dedicated opening authority) that owns a so-called *opening* key. The opening key allows the group manager, when given a signature, to recover the signer's identity. Originally, the opening was intended to prevent abuse of anonymity, and rather meant to be used in extreme situations. Clearly, the opening capability can also be leveraged to determine the linkability of various data events, but at high costs for privacy: every request for linkability will recover the full identity of the signer, and the central group manager learns the (signed) data of the data collectors and their correlation.

Group Signatures with Controlled Linkability. A more suitable solution are group signatures with controlled linkability [29, 30, 37]. In these schemes, signatures are unlinkable except to a dedicated linking authority with a secret key: on input

two signatures it tells whether they stem from the same user or not. This is much better then revealing the identity of the user, but still relies on a *fully trusted* entity that will learn the collectors' signed data. Further, this approach does not scale well for applications where a data collector is interested in the correlations within a large data set. To link a data set of n signed entries, each pair of signatures would have to be compared, which would require $n(n-1)/2$ requests to the linking authority. Another related concept are *traceable* group signatures [31] where a dedicated entity can generate a tracing trapdoor for each user which allows to trace this user's signatures. This approach is not suitable for our use case of controlled data linkage either, as it requires knowledge of the users' identities behind the anonymous group signatures or trapdoors for *all* users, and also needs every signature to be tested for every trapdoor.

Group Signatures with User-Controlled Linkability. Finally, schemes with user-controlled linkability exist, mostly in the context of Direct Anonymous Attestation (DAA) [11, 6, 12, 15, 14] or anonymous credentials [35, 19]. For linkability, a so-called *basename* is chosen alongside the message and all signatures with the same basename can easily be linked, but signatures for different basenames remain unlinkable. In contrast to solutions with opening or linking authorities, the linkability here can be publicly verified: a signature in such schemes contains a *pseudonym* that is deterministically derived from the user's secret key and the basename. Thus, the user re-uses the same pseudonym whenever he wants to be linkable. On the downside, this linkage is immediate and static. That is, the users have to choose at the beginning whether they want to disclose their data in a fully unlinkable manner, or linked w.r.t. a context-specific pseudonym. There is no option to selectively correlate the data after it has been disclosed. Therefore, users, or rather the data collectors allowing the use of such protocols, will hesitate to choose the option of unlinkability, as they fear to lose too much information by the irreversible decorrelation.

Our Contributions. In this work we overcome the aforementioned limitations by introducing a new type of group signatures that allows for flexible and selective linkability. We achieve that functionality by combining ideas from the different approaches discussed above: Group signatures are associated with pseudonyms, but pseudonyms are all *unlinkable per default*. Only when needed, a set of signatures – or rather the pseudonyms – can be linked in an efficient manner through a central entity, the *converter*. The converter receives a batch of pseudonymous data and transforms them into a consistent representation, meaning that all pseudonyms stemming from the same user will be converted into the same value. To preserve the privacy of the users and their data, the converter correlates the data in a fully blind way, i.e., not learning anything about the pseudonyms he transforms. We term these new form of group signatures CLS, which stands for convertably linkable (group) signatures.

Security and Privacy for CLS. A crucial property that we want from pseudonym conversions is that they establish linkability only strictly within the queried data,

i.e., linked pseudonyms from different queries should not be transitive. Otherwise, different re-linked data sets with overlapping input data could be pieced together, thereby gradually eroding the user's privacy. Aiming for such *non-transitivity* has an immediate impact on the overall setting: we need to channel both, the pseudonyms and messages, blindly through the converter, as transforming pseudonyms without the messages would require linkability between the in- and outputs of the conversion query, which in turn allows to correlate outputs from different queries.

We formally define the security of CLS through a number of security games, strongly inspired by the existing work on group signatures and DAA [4, 5, 15]. That is, we want signatures to be fully *anonymous* and unlinkable bearing in mind the information that is revealed through the selective linkability. We discuss that the classic anonymity notion adapted to our setting won't suffice, as it cannot guarantee the desired *non-transitivity*. In fact, capturing the achievable privacy and non-transitivity property in the presence of adaptive conversion queries was one of the core challenges of this work, and we formalize this property through a simulation-based definition. If the converter is corrupt, then unlinkability of signatures no longer holds, but the adversary should neither be able to trace signatures to a particular user, nor harm the obliviousness of queries which is captured in the *conversion blindness* and *join anonymity* properties. The guarantees in terms of unforgeability are captured through *non-frameability* and *traceability* requirements. The former says that corrupt users should not be able to impersonate honest users, and the latter guarantees that the power of an adversary should be bounded by the number of corrupt users he controls.

From a corruption point of view, we assume the data collector to be at most honest-but-curious towards the converter, i.e., even a corrupt data collector will only query pseudonym-message pairs that it has received along with a valid signature. We consider this a reasonable assumption, as data collectors that will use such a CLS scheme do so in order to implement the principle of data minimization on their own premises, and don't have an incentive to cheat themselves.

Efficient Instantiation. We propose an efficient construction of such CLS schemes, following the classical sign-and-encrypt paradigm that underlies most group signatures. Roughly, we use BBS+ signatures [3] for attesting group membership, i.e., a user will blindly receive a BBS+ signature from the group issuer on a secret key y chosen afresh by the user. To sign a message m on behalf of the group, the user computes a signature-proof-of-knowledge (SPK) for m where he proves knowledge of such an issuer's signature on its secret key and also encrypts it's user key (or rather its "public key" version h^y) under the converter's public key. The ciphertext that encrypts h^y serves as the pseudonym nym.

When the converter is asked to recover the correlations for a set of k pseudonym-message pairs $(nym_1, m_1), \ldots (nym_k, m_k)$, it blindly decrypts each pseudonym and blindly raises the result to the power of r which is chosen fresh for every conversion query, but used consistently within. That is, all pseudonyms belonging to the same user will be mapped to the same query-specific DDH tuple h^{yr} which allows for linkage of data within the query, but guarantees that converted

pseudonyms remain unlinkable across queries. To achieve obliviousness and non-transitivity of conversions, we encrypt all pseudonyms and messages with a re-randomisable (homomorphic) encryption scheme under the blinding key of the data collector. The re-randomisation is applied by the converter before he returns the transformed values, which ensures that the data collector cannot link the original and the converted pseudonyms by any cryptographic value. Clearly, if the associated messages are unique, then the data collector can link in- and outputs anyway, but our scheme should not introduce any additional linkage. Given that the pseudonyms are encryptions under the converter's public key, we need to add the second layer of encryption in a way that it doesn't interfere with the capabilities of the inner ciphertext. Using a nested form of ElGamal encryption [23] gives us these properties as well as the needed re-randomisability.

Finally, we prove that our instantiation satisfies the desired security and privacy requirements under the DDH, q-SDH and DCR assumption in the random oracle model. Our construction relies on type-3 pairings and performs most of the work in $\mathbb{G}_1$ which comes with significant efficiency benefits. In fact, we show that our construction is reasonably efficient considering the increased flexibility when establising the linkability in such a selective and controlled manner.

Other Related Work. A number of results exist that establish convertible pseudonyms in the setting of distributed databases and have inspired our work. Therein, the data gets created and maintained in a distributed manner. For privacy, related data is stored under different, database-specific pseudonyms that are seemingly unlinkable and can only be correlated by a central entity that controls the data flow. While the initial approach by Galindo and Verheul [26] required the converter to be a trusted third party, Camenisch and Lehmann [17, 18] later showed how the converter can operate in an oblivious manner. However, none of these solutions supports *authenticated* data collection and [26] and [17] even let the (trusted) converter establish all pseudonyms. The pseudonym system in [18] bootstraps pseudonyms in a blind way from a user secret, but for every new pseudonym that requires the user, converter and targeted data base to engage in an interactive protocol. Clearly, this is not practical for a setting where users frequently want to upload data. Further, all schemes re-use the same pseudonym for a user within a database, whereas our solution creates fresh and unlinkable pseudonyms for every new data item.

2 Preliminaries

This section presents all building blocks and assumptions that are needed for our CLS construction. We use ElGamal encryption as re-randomisable and homomorphic encryption scheme that is chosen plaintext secure, BBS+ signatures [3], and standard proof protocols.

Bilinear Maps & q-SDH Assumption. Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be cyclic groups of prime order p. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ must satisfy the following conditions:

bilinearity, i.e., $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$; *non-degeneracy*, i.e., for all generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1, g_2)$ generates $\mathbb{G}_T$; and *efficiency*, i.e., there exists an efficient algorithm $\mathcal{G}(1^\tau)$ that outputs a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, and an efficient algorithm to compute $e(a, b)$ for all $a \in \mathbb{G}_1$, $b \in \mathbb{G}_2$.

We use type-3 pairings [25] in this work, i.e., we do not assume $\mathbb{G}_1 = \mathbb{G}_2$ or the existence of an isomorphism between both groups in our scheme and security proofs. The advantage of type-3 pairings is that they enjoy the most efficient curves.

q-Strong Diffie Hellman Assumption (q-SDH). There are two versions of the q-Strong Diffie Hellman Assumption. The first version, given by Boneh and Boyen in [7], is defined in a type-1 or type-2 pairing setting. We use their second version of that definition that supports type-3 pairings and was stated in the journal version of their paper [8].

> Given $(g_1, g_1^\chi, g_1^{(\chi)^2}, ..., g_1^{(\chi)^q}, g_2, g_2^\chi)$ such that $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, output $(g_1^{\frac{1}{\chi+x}}, x) \in \mathbb{G}_1 \times \mathbb{Z}_p \backslash \{-\chi\}$.

BBS+ Signatures. Our scheme will make use of BBS+ signatures given by Au et al. [3], and inspired by BBS group signatures introduced in [9].

Key Generation: Take $(h_1, h_2) \leftarrow_\$ \mathbb{G}_1^2$, $x \leftarrow_\$ \mathbb{Z}_p^*$, $w \leftarrow g_2^x$, and set $sk = x$ and $pk = (w, h_1, h_2)$.

Signature: On input a message $m \in \mathbb{Z}_p$ and secret key x, pick $e, s \leftarrow_\$ \mathbb{Z}_p$ and compute $A \leftarrow (g_1 h_1^s h_2^m)^{\frac{1}{e+x}}$. Output signature $\sigma \leftarrow (A, e, s)$.

Verification: On input a public key $(w, h_1, h_2) \in \mathbb{G}_2 \times \mathbb{G}_1^2$, message $m \in \mathbb{Z}_p$, and purported signature $(A, e, s) \in \mathbb{G}_1 \times \mathbb{Z}_p^2$, check $e(A, w g_2^e) = e(g_1 h_1^s h_2^m, g_2)$.

When proving the unforgability of our scheme (called traceability in our setting), we will make use of techniques from [14] which prove the unforgeability of BBS+ signatures in the type-3 setting. Originally, Au et al. [3], proved the BBS+ signature secure under the first version of the q-SDH assumption given in [7], making use of the isomorphism between the groups in the security proof.

Re–Randomisable ElGamal Encryption. We use the ElGamal encryption scheme [23] with public parameters $(\mathbb{G}_1, g, p)$, such that the DDH problem is hard with respect to τ, i.e p is a τ bit prime.

Key Generation: Choose $sk \leftarrow_\$ \mathbb{Z}_p^*$, $pk \leftarrow g^{sk}$, and output (pk, sk).

Encryption: On input (pk, m), choose $r \leftarrow_\$ \mathbb{Z}_p^*$, and output $c \leftarrow (g^r, pk^r m)$.

Decryption: On input $(sk, (c_1, c_2))$, output $m \leftarrow c_2 c_1^{-sk}$.

ElGamal encryption is chosen-plaintext secure under the DDH assumption. In our construction, we will use the *homomorphic* property of ElGamal, i.e., if $C_1 \in \mathsf{Enc}(pk, m_1)$, and $C_2 \in \mathsf{Enc}(pk, m_2)$, then $C_1 \odot C_2 \in \mathsf{Enc}(pk, m_1 \cdot m_2)$.

We further use that ElGamal ciphertexts $c = \mathsf{Enc}(pk, m)$ are *publicly re-randomisable* in the sense that a re-randomised version c' of c looks indistinguishable from a fresh encryption of the underlying plaintext m. The following procedure clearly satisfies this:

Re-randomisation: On input $(pk, (c_1, c_2))$, get $r' \leftarrow_{\$} \mathbb{Z}_p^*$ and output $(c_1 g^{r'}, c_2 pk^{r'})$.

2.1 Proof Protocols

We follow the notation defined in [16] when referring to zero-knowledge proofs of knowledge of discrete logarithms. For example $\mathsf{PK}\{(a, b, c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$ denotes a zero knowledge proof of knowledge of integers a, b and c such that $y = g^a h^b$ and $\tilde{y} = \tilde{g}^a \tilde{h}^c$ hold. SPK denotes a signature proof of knowledge, that is a non-interactive transformation of a proof PK, e.g., using the Fiat-Shamir heuristic [24] in the random oracle. Using the Fiat-Shamir heuristic, the witness can be extracted from these proofs by rewinding the prover and programming the random oracle. Alternatively, these proofs can be extended to be online-extractable, by verifiably encrypting the witness to a public key defined in the common reference string. Clearly this requires a trusted common reference string. We underline the values that we need to be online-extractable in our proofs.

We require the proof system to be *simulation-sound* and *zero-knowledge*. The latter roughly says that there must exist a simulator that can generate simulated proofs which are indistinguishable from real proofs from the view of the adversary. The simulation-soundness is a strengthened version of normal soundness and guarantees that an adversary, even after having seen simulated proofs of false statements of his choice, cannot produce a valid proof of a false statement himself.

3 Definition & Security Model for CLS

In this section we first introduce the syntax and generic functionality of CLS and then present the desirable security and privacy properties for such schemes.

The following entities are involved in an CLS scheme: an issuer $\mathcal{I}$, a set of users $\mathcal{U} = \{uid_i\}$, a Verifier $\mathcal{V}$ and a converter $\mathcal{C}$. The issuer $\mathcal{I}$ is the central entity that allows users to join the group. Once joined, a user can then sign on behalf of the group in a pseudonymous way. That is, a verifier $\mathcal{V}$ can test the validity of a signature w.r.t the group's public key but does not learn any information about the particular user that created the signature. Most importantly, we want the pseudonymously signed data to be linkable in a controlled yet blind manner. Such selected linkability can be requested through the converter $\mathcal{C}$ that can blindly transform tuples of pseudonym-message pairs into a consistent representation.

3.1 Syntax of CLS

Our notation closely follows the definitional framework for dynamic group signatures given in [5]. We stress that our algorithms (and security notions) are flexible enough to cover settings where multiple verifiers and converters exist. For the sake of simplicity, however, we focus on the setting where there is only one entity each.

Definition 1 (CLS). *A convertably linkable group signature scheme CLS consists of the following algorithms:*

Setup & Key Generation. We model key generation individually per party, and refer to $(\mathsf{param}, ipk, cpk)$ as the group public key gpk.

Setup$(1^\tau) \to$ **param:** on input a security parameter 1^τ, outputs param, the public parameters for the scheme.

IKGen(**param**) $\to (ipk, isk)$**:** performed by the issuer $\mathcal{I}$, outputs the issuer secret key isk, and the issuing public key ipk.

CKGen(**param**) $\to (cpk, csk)$**:** performed by the Converter $\mathcal{C}$, outputs the converter secret key csk, and the converter public key cpk.

BKGen(**param**) $\to (bpk, bsk)$**:** performed by the verifier $\mathcal{V}$[3], outputs a blinding secret key, bsk, and blinding public key, bpk. As the key is only used for blinding purposes, (bpk, bsk) can be ephemeral. We write $\mathcal{BPK}$ as the public key space induced by BKGen.

Join, Sign & Verify. As in standard dynamic group signatures we have a dedicated join procedure that a user has to complete with the issuer. All users that have successfully joined the group can then create pseudonymous signatures on behalf of the group, i.e., that verify w.r.t. the group public key bpk. For ease of expression we treat the pseudonym nym as a dedicated part of the signature.

$\langle$**Join**(gpk), **Issue**$(isk, gpk)\rangle$**:** a user uid joins the group by engaging in a interactive protocol with the Issuer $\mathcal{I}$. The user uid and Issuer $\mathcal{I}$ perform algorithms Join and Issue respectively. These are input a state and an incoming message respectively, and output an updated state, an outgoing message, and a decision, either cont, accept, or reject. The initial input to Join is the group public key, gpk, whereas the initial input to Issue is the issuer secret key, isk, and the issuer public key ipk. If the user uid accepts, Join has a private output of $\mathbf{gsk}[uid]$.

Sign$(gpk, \mathbf{gsk}[uid], m) \to (nym, \sigma)$**:** performed by the user with identifier uid, with input the group public key gpk, the user's secret key $\mathbf{gsk}[uid]$, and a message m. Outputs a pseudonym nym and signature σ.

Verify$(gpk, m, nym, \sigma) \to \{0, 1\}$**:** performed by the Verifier $\mathcal{V}$. Outputs 1 if σ is a valid signature on m for pseudonym nym under the group public key gpk, and 0 otherwise.

Blind Conversion. Finally, we want our pseudonymous group signatures to be blindly convertable. Thus, we introduce a dedicated Blind and $\mathsf{Unblind}$ procedure for the verifier and a $\mathsf{Convert}$ algorithm that requires the converter's secret key. The latter transforms the unlinkable pseudonyms in a consistent manner, i.e., outputting converted pseudonyms that are consistent whenever the input pseudonyms belong to the same user.

[3] For sake of simplicity we state the algorithms for the setting where the requester and receiver of conversions is the same party, namely the verifier. However, our algorithms work in a public key setting to facilitate more general settings as well.

Blind$(gpk, bpk, (nym, m)) \rightarrow (cnym, c)$**:** performed by the verifier $\mathcal{V}$ on input a pseudonym-message pair(nym, m) and blinding public key bpk, group public key gpk. Outputs a blinded pseudonym and message.

Convert$(gpk, csk, bpk, \{(cnym_i, c_i)\}_k) \rightarrow \{(\overline{cnym}_i, \overline{c}_i)\}_k$**:** performed by the converter $\mathcal{C}$, on input k blinded pseudonym-message tuples $\{(cnym_i, c_i)\}_k = ((cnym_1, c_1), ..., (cnym_k, c_k))$, and the public blinding key bpk used. Outputs converted pseudonyms $\{(\overline{cnym}_i, \overline{c}_i)\}_k = ((\overline{cnym}_1, \overline{c}_1), ..., (\overline{cnym}_k, \overline{c}_k))$

Unblind$(bsk, (\overline{cnym}, \overline{c})) \rightarrow (\overline{nym}, \overline{m})$**:** performed by the Verifier $\mathcal{V}$ on input a converted pseudonym-message tuple, and the blinding secret key bsk. Outputs an unblinded converted pseudonym-message tuple $(\overline{nym}, \overline{m})$.

We sometimes make the randomness r used in these algorithms explicit, and e.g. write $\mathsf{Blind}(gpk, bpk, (nym, m); r)$.

3.2 Security Properties

We want that CLS schemes enjoy roughly the same security and privacy properties as group signatures when taking the added linkability into account. Defining these properties when pseudonyms can be *selectively* and *adaptively* converted is very challenging, though, as it requires a lot of care to avoid trivial wins while keeping the adversary as powerful as possible.

In a nutshell, we require the following guarantees from convertably linkable group signatures, where *(join) anonymity* and *non–transitivity* capture the privacy-related properties and *non–frameability* and *traceability* formalize the desired unforgeability.

(Join) Anonymity: Pseudonymous signatures should be unlinkable and untraceable (to a join session) even when the issuer and verifier are corrupt. When the converter is honest, unlinkability holds for all signatures for which the associated pseudonyms have not been explicitly linked through a conversion request. If the converter is corrupt and also controlled by the adversary, unlinkability is no longer possible, yet the anonymity of joins must remain.

Non–Transitivity: Converted pseudonyms should be non-transitive, i.e., the verifier should not be able to link the outputs of different convert queries. Otherwise, a corrupt verifier would be able to gradually link together all pseudonyms that have ever been queried to the converter.

Conversion Blindness: The converter learns nothing about the pseudonyms (and messages) it receives and the transformed pseudonyms it computes.

Non–Frameability: An adversary controlling the issuer and some corrupt users, should not be able to impersonate other honest users, i.e., create pseudonymous signatures that would be linked to a pseudonym of an honest user.

Traceability: An adversary should not be able to create more signatures that remain unlinkable in a conversion than he controls corrupt users.

Clearly, any re-linked subset of the originally anonymous data increases the risk of re-identification. Thus, the converter could enforce some form of

access control to the re-linked data, e.g., only converting a certain amount of pseudonyms at once. The non-transitivity requirement then ensures that a corrupt verifier cannot further aggregate the individually learned data. We stress that our security properties only formalize the achievable privacy for the pseudonyms and signatures. They do not *and cannot* capture information leakage through the messages that the users sign. This is the case for all group signatures though, and not special to our setting.

Oracles & State. The security notions we formalize in the following make use a number of oracles which keep joint state, e.g., keeping track of queries and the set of corrupted parties. We present the detailed description of all oracles in Fig. 1 and an overview of them and their maintained records below.

ADDU (join of honest user & honest issuer) Creates a new honest user for *uid* and internally runs a join protocol between the honest user and honest issuer. At the end, the honest user's secret key $\mathbf{gsk}[uid]$ is generated and from then on signing queries for *uid* will be allowed.

SNDU (join of honest user & corrupt issuer) Creates a new honest user for *uid* and runs the join protocol on behalf of *uid* with the corrupt issuer. If the join session completes, the oracle will store the user's secret key $\mathbf{gsk}[uid]$.

SNDI (join of corrupt user & honest issuer) Runs the join protocol on behalf of the honest issuer with corrupt users. For joins of honest users, the ADDU oracle must be used.

SIGN This oracle returns signatures for honest users that have successfully joined (via ADDU or SNDU, depending on the game).

CONVERT The oracle returns a set of converted pseudonyms along with their messages. To model that conversion is triggered by an at most honest-but-curious verifier, we request $\mathcal{V}$ to provide the unblinded set of pseudonyms along with signatures. The conversion will only be done when all signatures are valid. The oracle then internally blinds the pseudonym-message pairs and returns the blinded input, the randomness used for the blinding along with the converted output. When this oracle is used in the anonymity game, it further checks that the input does not allow the adversary to trivially win by converting the challenge pseudonym together with pseudonyms from either of the challenge users.

All oracles have access to the following records maintained as global state:

HUL List of *uid*'s of honest users, initially set to $\varnothing$. New honest users can be added by queries to the ADDU oracle (when the issuer is honest) or SNDU oracle (when the issuer is corrupt).

CUL List of corrupt users that have (requested) to join the group. Initially set to $\varnothing$, new corrupt users can be added through the SNDI oracle if the issuer is honest. If the issuer is corrupt, we do not keep track of corrupt users.

SL List of (uid, m, nym, σ) tuples requested from the SIGN oracle.

ADDU(uid)

if $uid \in \mathsf{HUL} \cup \mathsf{CUL}$ **return** $\perp$
$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp$
$\mathbf{dec}^{uid} \leftarrow \mathsf{cont}, \mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk)$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow \mathsf{Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, \perp)$
while $\mathbf{dec}^{uid} = \mathsf{cont}$
  $(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Join}}, \mathbf{dec}^{uid}) \leftarrow \mathsf{Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{Issue}})$
  $(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Issue}}, \mathbf{dec}^{uid}) \leftarrow \mathsf{Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Join}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return accept

SIGN(uid, m)

if $uid \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid] = \perp$
  return $\perp$
$(nym, \sigma) \leftarrow \mathsf{Sign}(gpk, \mathbf{gsk}[uid], m)$
$\mathsf{SL} \leftarrow \mathsf{SL} \cup \{(uid, m, nym, \sigma)\}$
return (σ, nym)

SNDI(uid, M_{in})

if $uid \in \mathsf{HUL}$ **return** $\perp$
if $uid \notin \mathsf{CUL}$
  $\mathsf{CUL} \leftarrow \mathsf{CUL} \cup \{uid\}$
  $dec^{uid} \leftarrow \mathsf{cont}$
if $dec^{uid} \neq \mathsf{cont}$ **return** $\perp$
if undefined $\mathsf{st}^{uid}_{\mathsf{Issue}} \leftarrow (isk, gpk)$
$(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{out}}, \mathbf{dec}^{uid}) \leftarrow \mathsf{Issue}(\mathsf{st}^{uid}_{\mathsf{Issue}}, M_{\mathsf{in}})$
return $(M_{\mathsf{out}}, \mathsf{dec}^{uid})$

SNDU(uid, M_{in})

if $uid \in \mathsf{CUL}$ **return** $\perp$
if $uid \notin \mathsf{HUL}$
  $\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}$
  $\mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$
if $\mathbf{dec}^{uid} \neq \mathsf{cont}$ **return** $\perp$
if $\mathsf{st}^{uid}_{\mathsf{Join}}$ undefined $\mathsf{st}^{uid}_{\mathsf{Join}} \leftarrow gpk$
$(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{Out}}, \mathbf{dec}^{uid}) \leftarrow \mathsf{Join}(\mathsf{st}^{uid}_{\mathsf{Join}}, M_{\mathsf{in}})$
if $\mathbf{dec}^{uid} = \mathsf{accept}$ $\mathbf{gsk}[uid] \leftarrow \mathsf{st}^{uid}_{\mathsf{Join}}$
return $(M_{\mathsf{Out}}, \mathbf{dec}^{uid})$

CONVERT($(nym_1, m_1, \sigma_1), \ldots, (nym_k, m_k, \sigma_k), bpk$)

if $\exists i \in [1, k]$ s.t. $\mathsf{Verify}(gpk, m_i, nym_i, \sigma_i) = 0$ **return** $\perp$
if $bpk \notin \mathcal{BPK}$ **return** $\perp$
if $\exists i$ s.t. $nym_i = nym^*$ and $\exists j \neq i$ s.t. $\mathsf{Identify}(uid^*_d, nym_j) = 1$ for $d \in \{0, 1\}$
  return $\perp$
else compute $(cnym_i, c_i) \leftarrow \mathsf{Blind}(gpk, bpk, (nym_i, m_i); r_i)$ for $i = 1, \ldots, k$
  and $\{(\overline{cnym}_i, \overline{c}_i)\}_k \leftarrow \mathsf{Convert}(gpk, csk, bpk, \{(cnym_i, c_i)\}_k)$
return $(\{(cnym_i, c_i)\}_k, \{(\overline{cnym}_i, \overline{c}_i)\}_k, r_1, \ldots, r_k)$

Fig. 1. Oracles used in our security games

Helper Algorithms. We introduce two additional algorithms for notational simplicity in our security games: Identify and UnLink. Roughly, Identify allows to test whether a pseudonym belongs to a certain *uid* by exploiting the convertability of pseudonyms. That is, we create a second signature for $\mathbf{gsk}[uid]$ and use the converter's secret key to test whether both are linked. If so, Identify returns 1. This algorithm already uses our second helper algorithm UnLink internally, which takes a list of (correctly formed) pseudonym-message pairs and returns 1 if they are all unlinkable and 0 otherwise.

$\mathsf{Identify}(gpk, csk, uid, m, nym)$

$(nym', \sigma') \leftarrow \mathsf{Sign}(gpk, \mathbf{gsk}[uid], 0)$
if $\mathsf{UnLink}(gpk, csk, ((nym, m), (nym', 0))) = 0$ **return** 1
else return 0

$\mathsf{UnLink}(gpk, csk, ((nym_1, m_1), ..., (nym_k, m_k)))$

$(bpk, bsk) \leftarrow \mathsf{BKGen}(\mathsf{param})$
$\forall i \in [1, k] \quad (cnym_i, c_i) \leftarrow \mathsf{Blind}(gpk, bpk, (nym_i, m_i))$
$\{(\overline{cnym}_i, \overline{c}_i)\}_k \leftarrow \mathsf{Convert}(gpk, csk, bpk, \{(cnym_i, c_i)\}_k)$
$\forall i \in [1, k] \quad (\overline{nym}_i, \overline{m}_i) \leftarrow \mathsf{Unblind}(bsk, (\overline{cnym}_i, \overline{c}_i))$
if $\exists (i, j)$ with $i \neq j$ s.t. $\overline{nym}_i = \overline{nym}_j$ **return** 0
else return 1

For even more simplicity we often omit the keys for the algorithms (as they are clear from the context). That is, we write $\mathsf{Identify}(uid, nym)$ which will indicate whether the pseudonym *nym* belongs to the user with identity *uid* or not. Likewise we write $\mathsf{UnLink}(nym_1, \ldots, nym_k)$ to test whether all pseudonyms are uncorrelated or not.

Correctness. CLS signatures should be correct and consistent when being produced by honest parties. More precisely, we formulate correctness via three requirements: *Correctness of sign* guarantees that signatures formed using the Sign algorithm with a user secret key generated honestly will verify correctly. *Correctness of conversion* guarantees that after blinding, converting and then unblinding correctly, the output will be correctly linked messages/ pseudonyms. *Consistency* is a stronger variant of conversion-correctness and requires that the correlations of pseudonyms established through the conversion procedure must be consistent across queries. More precisely, if a conversion query reveals that two pseudonym nym_1 and nym_2 are linked, and another one that nym_2 and nym_3 are linked, then it must also hold that a conversion query for nym_1 and nym_3 returns linked pseudonyms. We require that this property even holds for maliciously formed pseudonyms, which will be a helpful property in some of our security proofs. For space reasons, the detailed correctness definitions are deferred to the full paper [27].

Anonymity *(Corrupt Issuer & Verifier).* This security requirement captures the desired anonymity properties when both the issuer and verifier are corrupt. Just as in conventional group signatures, we want that the signatures of honest users are unlinkable and cannot be traced back to a user's join session with the corrupt issuer. To model this property, we let the adversary output *uid*'s of two honest users together with a message and return a challenge (nym^*, σ^*) that is created either by user uid_0 or uid_1. For anonymity, the adversary should not be able to determine the user's identity better than by guessing.

In our setting, this property must hold when the corrupt verifier has access to the conversion oracle where it can obtain linked subsets of the pseudonymous data. To avoid trivial wins, the adversary is not allowed to make conversion queries that link the challenge pseudonym nym^* to another pseudonym belonging to one of the two honest challenge users.

Definition 2 (Anonymity). *A CLS scheme satisfies anonymity if for all polynomial time adversaries $\mathcal{A}$ the following advantage is negligible in τ :*

$$\left|\Pr[\mathbf{Exp}^{anon-0}_{\mathcal{A},CLS}(\tau) = 1] - \Pr[\mathbf{Exp}^{anon-1}_{\mathcal{A},CLS}(\tau) = 1]\right|.$$

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

$\mathsf{param} \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(\mathsf{param}), (cpk, csk) \leftarrow \mathsf{CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \emptyset$
$(uid_0^*, uid_1^*, m^*, st) \leftarrow \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(\mathsf{choose}, gpk, isk)$
if $uid_0^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0^*] = \perp$ or $uid_1^* \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_1^*] = \perp$ **return** 0
$(nym^*, \sigma^*) \leftarrow \mathsf{Sign}(gpk, \mathbf{gsk}[uid_b^*], m^*)$
$b^* \leftarrow \mathcal{A}^{\mathsf{SNDU,SIGN,CONVERT}}(\mathsf{guess}, st, nym^*, \sigma^*)$
return b^*

Non–Transitivity *(Corrupt Issuer & Verifier).* The second privacy-related property we want to guarantee when both the issuer and verifier are corrupt, is the strict non-transitivity of conversions. This ensures that the outputs of separate convert queries cannot be linked together, further than what is already possible due the messages queried. For example if $\overline{nym}_1$ and $\overline{nym}_2$ are outputs by two separate convert queries, the adversary should not be able to decide whether they were derived from the same pseudonym or not. Otherwise the verifier could gradually build lists of linked pseudonyms, adding to these during every convert query and eventually recover the linkability among all pseudonymous signatures.

To model non-transitivity of conversions we use a simulation-based approach, requiring the indistinguishability of an ideal and a real world. In the real world, all convert queries are handled normally through the CONVERT oracle defined in Fig 1. Whereas in the ideal world, the converted pseudonyms are produced by a simulator SIM through the CONVSIM oracle defined below. For a conversion request of input $(nym_1, m_1, \sigma_1), \ldots, (nym_k, m_k, \sigma_k)$ the simulator will only learn which of the messages belong together, i.e., are associated to pseudonyms that belong to the same user *uid*. For *honest* users this can be looked up through

the records of the signing oracle that stores tuples $(uid, m_i, nym_i, \sigma_i)$ for each signing query. Thus, we let the simulator mimic the conversion output for all pseudonyms stemming form honest users, and convert pseudonyms from corrupt users normally (as there is no privacy to guarantee for them anyway). Finally, the CONVSIM oracle outputs a random shuffle of the correctly converted pseudonyms of corrupt users, and the simulated ones for honest users. As mentioned before, we assume the verifier to be honest-but-curious, which we enforce by requesting the adversary to output valid signatures along with the pseudonyms to be converted and handle the blinding step within the conversion oracle.

Definition 3 (Non-Transitivity). *A CLS scheme satisfies non-transitivity if for all polynomial time adversaries $\mathcal{A}$ there exists an efficient simulator SIM such that the following advantage is negligible in τ:*

$$\left|\Pr[\mathbf{Exp}^{nontrans-0}_{\mathcal{A},CLS}(\tau) = 1] - \Pr[\mathbf{Exp}^{nontrans-1}_{\mathcal{A},CLS}(\tau) = 1]\right|.$$

Experiment: $\mathbf{Exp}^{nontrans-b}_{\mathcal{A},\mathsf{CLS}}(\tau)$

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

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \varnothing$

$b^* \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(\mathsf{guess}, gpk, isk)$

where the oracle CONVX works as follows:

if $b = 0$ (real world) **then** CONVX is the standard CONVERT oracle

if $b = 1$ (ideal world) **then** CONVX is the simulated CONVSIM oracle

return b^*

$\mathsf{CONVSIM}((nym_1, m_1, \sigma_1), \ldots, (nym_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1, k]$ s.t. $\mathsf{Verify}(gpk, m_i, nym_i, \sigma_i) = 0$ or $bpk \notin \mathcal{BPK}$ **return** $\perp$

Set $\mathsf{CL} \leftarrow \varnothing$

Compute $(cnym_i, c_i) \leftarrow \mathsf{Blind}(gpk, bpk, (nym_i, m_i); r_i)$ for $i = 1, \ldots, k$

$\forall i \in [1, k]$ // determine message clusters L_{uid} for honest users and list CL of corrupt pseudonyms

if $(uid, m_i, nym_i, \sigma_i) \in \mathsf{SL}$ // pseudonyms from honest users

if L_{uid} does not exist, create $\mathsf{L}_{uid} \leftarrow \{m_i\}$ **else** set $\mathsf{L}_{uid} \leftarrow \mathsf{L}_{uid} \cup \{m_i\}$

else $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, cnym_i)\}$ // pseudonyms from corrupt users

$\{(\overline{cnym}_i, \bar{c}_i)\}_{i=1,\ldots k'} \leftarrow \mathsf{Convert}(gpk, csk, bpk, \mathsf{CL})$ for $k' \leftarrow |\mathsf{CL}|$ // normally convert corrupt nyms

Let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters created above

$\{(\overline{cnym}_i, \bar{c}_i)\}_{i=k'+1,\ldots k} \leftarrow \mathsf{SIM}(gpk, bpk, \mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}})$ // simulate conversion for honest nyms

Let $\{(\overline{cnym}'_i, \bar{c}'_i)\}_{i=1,\ldots k}$ be a random permutation of $\{(\overline{cnym}_i, \bar{c}_i)\}_{i=1,\ldots k}$

return $(\{(cnym_i, c_i, r_i)\}_{i=1,\ldots,k}, \{(\overline{cnym}'_i, \bar{c}'_i)\}_{i=1,\ldots k}, r_1, \ldots, r_k)$

Anonymity vs. Non–Transitivity. Note that non-transitivity is not covered by the anonymity notion defined before: A scheme that satisfies anonymity could output the converted pseudonyms in the exact same order as the input ones, allowing trivial linkage between the in- and output of each conversion request. Thus, whenever the same pseudonym is used as input to several conversion queries, this would enable the linkability of the transformed pseudonyms across the different conversions, which is exactly what non-transitivity aims to avoid.

On the first glance, it might seem odd that having transitive conversions does not harm our anonymity property. However, transitivity is only useful when *several* pseudonyms belonging to the same user appear in each conversion request with one pseudonym being re-used in all these sessions. In the anonymity game, the challenge pseudonym is not allowed to be used in combination with any other pseudonym stemming from either of the challenge users (as this would make the definition unachievable), and thus the transitivity of conversions can not be exploited.

Conversion Blindness *(Corrupt Issuer & Converter).* A crucial property of our signatures is that they can be converted in an oblivious manner, i.e., without the converter learning anything about the pseudonyms or messages it converts. In particular, this blindness property ensures the unlinkability of blinded inputs across several conversion requests. Conversion blindness should hold if both the issuer and converter are corrupt, but the verifier is honest. We formalize this property in a classic indistinguishability style: the adversary outputs two tuples of pseudonym-message pairs and receives a blinded version of either of them. Given that blinding of pseudonyms is a public-key operation we don't need an additional blinding oracle. In fact, we don't give the adversary *any* oracle access at all in this game. He already corrupts the issuer and converter, and this property does not distinguish between honest and corrupt users, thus we simply assume that the adversary has full control over all users as well.

Definition 4 (Conversion Blindness). *A CLS scheme satisfies conversion blindness if: for all polynomial time adversaries $\mathcal{A}$ the following advantage is negligible in τ:*

$$\left|\Pr[\mathbf{Exp}^{blind-conv-0}_{\mathcal{A},\mathsf{CLS}}(\tau)=1]-\Pr[\mathbf{Exp}^{blind-conv-1}_{\mathcal{A},\mathsf{CLS}}(\tau)=1]\right|.$$

Experiment: $\mathbf{Exp}^{blind-conv-b}_{\mathcal{A},\mathsf{CLS}}(\tau)$

$\mathsf{param} \leftarrow \mathsf{Setup}(1^\tau), (ipk, isk) \leftarrow \mathsf{IKGen}(\mathsf{param}), (cpk, csk) \leftarrow \mathsf{CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk),\ \ (bpk, bsk) \leftarrow \mathsf{BKGen}(\mathsf{param})$
$(st, (nym_0, m_0), (nym_1, m_1)) \leftarrow \mathcal{A}(\mathsf{choose}, gpk, isk, csk, bpk)$
$(cnym^*, c^*) \leftarrow \mathsf{Blind}(gpk, bpk, (nym_b, m_b))$
$b^* \leftarrow \mathcal{A}(\mathsf{guess}, st, cnym^*, c^*)$
return b^*

Join Anonymity *(Corrupt Issuer & Converter & Verifier).* The final privacy related property we require from a CLS is the anonymity of joins even if *all* central entities are corrupt. Here the challenge is that the adversary, controlling the issuer, converter and verifier, should not be able to link signatures of an honest user back to a particular join session. This is the best one can hope for in this corruption setting as unlinkability of signatures (as guaranteed by our anonymity property) is no longer possible: the corrupt converter can simply convert all signatures/pseudonyms into a consistent representation. Such a

property does not exist in conventional group signatures, as therein a corrupt opener can always reveals the join identity. In our setting, signatures can only be linked instead of being opened and thus anonymity of the join procedure can and should be preserved.

To model this property we let the adversary output two identities of honest users uid_0, uid_1 that have successfully joined. We then give the adversary access to a signing oracle for one them. This is done by adding the challenge user uid^* (where uid^* stands for a dummy handle) to the list of honest users HUL with user secret key $\mathbf{gsk}[uid_b]$. Thus, in the second stage of the game, the adversary can invoke the SIGN oracle on uid^* to receive signatures of messages of his choice for the challenge user. The adversary wins if he can determine the bit b better than by guessing. To avoid trivial wins, the adversary is not allowed to see any signature directly from uid_0 or uid_1.

Definition 5 (Join Anonymity). *A CLS scheme satisfies join anonymity if: for all polynomial time adversaries $\mathcal{A}$ the following advantage is negligible in τ:*

$$\left|\Pr[\mathbf{Exp}^{anon-join-0}_{\mathcal{A},CLS}(\tau)=1]-\Pr[\mathbf{Exp}^{anon-join-1}_{\mathcal{A},CLS}(\tau)=1]\right|.$$

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

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

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \varnothing$

$(st, uid_0, uid_1) \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(\mathsf{choose}, gpk, isk, csk)$

if uid_0 or $uid_1 \notin \mathsf{HUL}$ or $\mathbf{gsk}[uid_0] = \perp$ or $\mathbf{gsk}[uid_1] = \perp$ **return** $\perp$

Choose $uid^*, \mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid^*\}, \mathbf{gsk}[uid^*] \leftarrow \mathbf{gsk}[uid_b]$

$b^* \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(\mathsf{guess}, st, uid^*)$

return b^* **if** $(uid_d, *) \notin \mathsf{SL}$ for $d = 0, 1$ **else return** 0

Non–Frameability ***(Corrupt Issuer & Converter & User)*** This notion captures the desired unforgeability properties when the issuer, converter and some of the users are corrupt, and requires that an adversary should not be able to impersonate an honest user. Our definition is very similar to the non-frameability definitions in standard group signature or DAA schemes [4–6]. Roughly, the only part we have to change is how we detect that an honest user has been framed. In group signatures, non-frameability exploits the presence of the group manager that can open signatures and requests that an adversary cannot produce signatures that will open to an honest user who hasn't created said signature. Here we have the converter instead of the group manager (or dedicated opening authority), and thus express non-frameability through the linkage that is created in a conversion. More precisely, an adversary should not be able to produce a valid signature (nym^*, σ^*) that within a conversion request would falsely link to a signature of an honest user. For generality (and sake of brevity), we use our helper function $\mathsf{Identify}$ that we introduced at the beginning of this section to express that the adversary's signature should not be recognized as a signature of an honest user.

Definition 6 (Non-Frameability). *A CLS scheme satisfies non–frameability if for all polynomial time adversaries $\mathcal{A}$, the advantage* $\Pr[\mathbf{Exp}^{nonframe}_{\mathcal{A},CLS}(\tau) = 1]$ *is negligible in* τ.

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

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

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \varnothing$

$(uid, m^*, nym^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN}}(gpk, isk, csk)$

return 1 **if** all of the following conditions are satisfied:

$\mathsf{Verify}(gpk, m^*, nym^*, \sigma^*) = 1$ and

$\mathsf{Identify}(uid, m^*, nym^*) = 1$ where $uid \in \mathsf{HUL}$ and

$(uid, m^*, nym^*, \sigma^*) \notin \mathsf{SL}$

Traceability ***(Corrupt Converter & User)*** Our final requirement formalizes the unforgeability properties when only the converter and some users are corrupt. In this setting, an adversary should not be able to output more pseudonymous signatures that remain unlinkable in a conversion than the number of users it has corrupted. This is again an adaptation of the existing traceability notions for group signatures with an opening authority [4, 5] or user-controlled linkability [6]. Interestingly, in the latter work (that is closer to our setting than standard group signatures), two traceability notions where introduced: While one is similar in spirit to our notion, a second property guarantees that all signatures of corrupt users can be traced back to the exact signing key that the corrupt user has established in the join protocol with the honest issuer. This seems a bit of an odd requirement, as it is not noticeable in the real world. In fact, we do not limit the strategy of the adversary in that way and only require his power to be bounded by the amount of corrupt users he controls.

Our definition stated below uses our helper algorithm UnLink that we introduced at the beginning of this section and that internally uses the Convert algorithm to detect whether pseudonyms are unlinkable or not. Note that UnLink returns 1 only if *all* inputs are mutually unlinkable, i.e., there is not a single tuple of input pseudonyms that got converted to the same value.

Definition 7 (Traceability). *A CLS scheme satisfies traceability if for all polytime adversaries $\mathcal{A}$ the advantage* $\Pr[\mathbf{Exp}^{trace}_{\mathcal{A},CLS}(\tau) = 1]$ *is negligible in* τ.

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

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

$gpk \leftarrow (\mathsf{param}, ipk, cpk), \mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \varnothing$

$((m_1, nym_1, \sigma_1), ..., (m_k, nym_k, \sigma_k)) \leftarrow \mathcal{A}^{\mathsf{ADDU},\mathsf{SNDI},\mathsf{SIGN}}(gpk, csk)$

return 1 if all of the following conditions are satisfied:

$\forall j \in [1, k] : \mathsf{Verify}(gpk, m_j, nym_j, \sigma_j) = 1$ and $(*, m_j, nym_j, \sigma_j) \notin \mathsf{SL}$ and

$k > |\mathsf{CUL}|$ and

$\mathsf{UnLink}(gpk, csk, ((nym_1, m_1), ..., (nym_k, m_k))) = 1$

4 Our CLS Construction

We now present our construction that securely realizes such CLS group signatures. Our scheme follows the classical sign-and-encrypt paradigm: we use BBS+ signatures [3] for attesting group membership, i.e., a user will blindly receive a BBS+ signature from the group issuer on the user's secret key y. To sign a message m on behalf of the group, the user computes a SPK for m where he proves knowledge of a BBS+ signature on y and also encrypts h^y under the converter's public key. The ElGamal ciphertext that encrypts h^y serves as the associated pseudonym nym.

To blind a set of k pseudonym-message pairs $(nym_1, m_1), \ldots (nym_k, m_k)$ for conversion, the verifier encrypts each value under its own ElGamal public key. As the pseudonyms are already ElGamal ciphertexts themselves, this results in a nested double-encryption of h^y being encrypted under both keys. The converter then decrypts the "inner" part of the ciphertext and blindly raises the result to a random value r. This r is chosen fresh for every conversion query, but used consistently within. That is, all pseudonyms belonging to the same user will be mapped to the same query-specific DDH tuple h^{yr}. Finally, the converter re-randomises all ciphertexts and shuffles them to destroy any linkage between the in- and output tuples — this is crucial for achieving the desired non-transitivity property. The verifier then simply decrypts the received tuples and can link correlated data via the converted pseudonyms $\overline{cnym}_i$.

4.1 Detailed Description of CLS–DDH

Setup & Key Generation. We use a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ with g_1 and g_2 being generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Further, we need four additional generators g, h and h_1, h_2 in $\mathbb{G}_1$, where h_1, h_2 are used for the BBS+ part, and g, h will be used for the ElGamal encryption.

$\underline{\mathsf{Setup}(1^\tau)}$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^\tau), g, h, h_1, h_2 \leftarrow_\$ \mathbb{G}_1$
return $\mathsf{param} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, g_2, g, h, h_1, h_2)$

$\underline{\mathsf{IKGen}(\mathsf{param})}$	$\underline{\mathsf{CKGen}(\mathsf{param})}$	$\underline{\mathsf{BKGen}(\mathsf{param})}$
$isk \leftarrow_\$ \mathbb{Z}_p^*, ipk \leftarrow g_2^{isk}$	$csk \leftarrow_\$ \mathbb{Z}_p^*, cpk \leftarrow g^{csk}$	$bsk \leftarrow_\$ \mathbb{Z}_p^*, bpk \leftarrow g^{bsk}$
return (ipk, isk)	**return** (cpk, csk)	**return** (bpk, bsk)

Join. To join the group, users perform an interactive protocol with the issuer to obtain their user secret key and group credential. Roughly, the $\mathbf{gsk}[uid]$ of a user consists of a secret key $y \in \mathbb{Z}_p^*$ and a BBS+ signature (A, x, s) of $\mathcal{I}$ on y, where $A = (g_1 h_1^y h_2^s)^{1/(isk+x)}$. The detailed protocol of $\langle \mathsf{Join}(gpk), \mathsf{Issue}(isk, gpk) \rangle$ is given in Figure 2.

$\mathcal{U}.\mathsf{Join}(gpk)$	$\rightleftharpoons$	$\mathcal{I}.\mathsf{Issue}(isk, gpk)$
		choose $n \leftarrow_\$ \{0,1\}^\tau$
	$\xleftarrow{n}$	
$y \leftarrow_\$ \mathbb{Z}_p^*, H \leftarrow h_1^y,$		
$\pi_H \leftarrow \mathsf{SPK}\{(\underline{y}) : H = h_1^y\}(n)$		
	$\xrightarrow{H, \pi_H}$	
		Verify π_H and that $H \in \mathbb{G}_1$
		$x \leftarrow_\$ \mathbb{Z}_p^*, s \leftarrow_\$ \mathbb{Z}_p^*$
		$A \leftarrow (H h_2^s g_1)^{1/(isk+x)}$
	$\xleftarrow{A, x, s}$	
check that $A \neq 1_{\mathbb{G}_1}$ and that		
$e(A, g_2)^x e(A, ipk) = e(g_1 H h_2^s, g_2)$		
return $\mathbf{gsk}[uid] \leftarrow (A, x, y, s)$		

Fig. 2. Join protocol of our CLS–DDH construction.

Sign & Verify. To sign a message m under $\mathbf{gsk}[uid] = (A, x, y, s)$, the user proves knowledge of a BBS+ credential (A, x, s) on its secret key y and also encrypts h^y under the converter's public key cpk. The proof π then proves knowledge of the BBS+ credential and asserts that the encryption contains the same value y. From π we only need the value y to be online extractable. We use the improved SPK from Camenisch et al. [14] who have shown how to move most of the work from $\mathbb{G}_T$ to $\mathbb{G}_1$ and thus yield a much faster instantiation than the original proof by Au et al. [3]. For verification, one verifies the proof π and some correctness properties of the re-randomised versions of A that are sent along with the proof. In more detail, Sign and Verify are defined as follows:

$\mathsf{Sign}(gpk, \mathbf{gsk}[uid], m)$

parse $\mathbf{gsk}[uid] = (A, x, y, s)$, $gpk = (ipk, cpk)$
$\alpha \leftarrow_\$ \mathbb{Z}_p^*,\ nym_1 \leftarrow g^\alpha,\ nym_2 \leftarrow cpk^\alpha h^y,\ r_1, r_2, r_3 \leftarrow_\$ \mathbb{Z}_p^*,$
$A' \leftarrow A^{r_1}, \hat{A} \leftarrow A'^{-x}(g_1 h_1^y h_2^s)^{r_1}, d \leftarrow (g_1 h_1^y h_2^s)^{r_1} h_2^{-r_2}, r_3 \leftarrow r_1^{-1}, s' \leftarrow s - r_2 r_3$
$\pi \leftarrow \mathsf{SPK}\{(x, \underline{y}, r_2, r_3, s', \alpha) : nym_1 = g^\alpha \wedge nym_2 = cpk^\alpha h^y$
$\quad \wedge \hat{A}/d = A'^{-x} h_2^{r_2} \wedge g_1 h_1^y = d^{r_3} h_2^{-s'}\}(m)$
$\sigma \leftarrow (A', \hat{A}, d, \pi), nym \leftarrow (nym_1, nym_2)$
return (nym, σ)

$\mathsf{Verify}(gpk, m, nym, \sigma)$

parse $\sigma = (A', \hat{A}, d, \pi)$
return 1 **if** $A' \neq 1_{\mathbb{G}_1}, e(A', ipk) = e(\hat{A}, g_2)$,
$\quad$ and π holds for $A', \hat{A}, d, nym, m$ w.r.t. gpk

Blind Conversions. When the verifier wants to learn which of the pseudonymously received messages belong together, it sends a batch of pseudonym-message pairs in blinded form to the converter. That is, it encrypts the messages and pseudonyms using ElGamal encryption. The pseudonyms are ElGamal ciphertexts itself and we roughly wrap them in another encryption layer. The converter then blindly decrypts the pseudonyms, i.e., decrypts the "inner" part of the ciphertext, which yields h^y encrypted under the verifiers blinding key bpk. To ensure non-transitivity, i.e., restrict the linkage of pseudonyms to hold only within the queried batch, the converter blindly raises the encrypted h^y to a random exponent r. This value is chosen afresh for every batch but used consistently within the query, i.e., all pseudonyms that belong to the same user with secret key y will be mapped consistently to h^{yr}. To ensure that the ciphertexts and their order cannot leak any additional information, we let the converter re-randomize and shuffle all ciphertexts before he returns them to the verifier. Both the verifier and the converter are assumed to be at most honest-but-curious, and so proofs that they have performed Blind and Convert correctly are not needed.

$\mathsf{Blind}(gpk, bpk, nym, m)$

parse $gpk = (ipk, cpk),\ nym = (nym_1, nym_2)$
$\alpha, \beta, \gamma \leftarrow_\$ \mathbb{Z}_p^*$
$cnym_1 \leftarrow nym_1 g^\beta,\ cnym_2 \leftarrow g^\alpha,\ cnym_3 \leftarrow nym_2 cpk^\beta bpk^\alpha$
$c_1 \leftarrow g^\gamma, c_2 \leftarrow bpk^\gamma m$
$cnym \leftarrow (cnym_1, cnym_2, cnym_3),\ c \leftarrow (c_1, c_2)$
return $(cnym, c)$

$\mathsf{Convert}(gpk, csk, bpk, ((cnym_1, c_1), ..., (cnym_k, c_k)))$

parse $cnym_i = (cnym_{i,1}, cnym_{i,2}, cnym_{i,3}),\ c_i \leftarrow (c_{i,1}, c_{i,2}),\ r \leftarrow_\$ \mathbb{Z}_p^*$
for $i = 1, \ldots k$:
  $cnym'_{i,1} \leftarrow cnym_{i,2}^r,\ cnym'_{i,2} \leftarrow (cnym_{i,3} cnym_{i,1}^{-csk})^r$ // decrypt nym and raise to r
  $r_1, r_2 \leftarrow_\$ \mathbb{Z}_p^*$ // re-randomize all ciphertexts
  $cnym''_{i,1} \leftarrow cnym'_{i,1} g^{r_1},\ cnym''_{i,2} \leftarrow cnym'_{i,2} bpk^{r_1}$
  $c'_{i,1} \leftarrow c_{i,1} g^{r_2},\ c'_{i,2} \leftarrow c_{i,2} bpk^{r_2}$
choose random permutation Π, **for** $i = 1, \ldots, k$: $(\overline{cnym}_i, \overline{c}_i) \leftarrow (cnym''_{\Pi(i)}, c'_{\Pi(i)})$
return $((\overline{cnym}_1, \overline{c}_1), ..., (\overline{cnym}_k, \overline{c}_k))$

$\mathsf{Unblind}(bsk, (\overline{cnym}, \overline{c}))$

parse $\overline{cnym} = (\overline{cnym}_1, \overline{cnym}_2),\ \overline{c} \leftarrow (\overline{c}_1, \overline{c}_2)$
$\overline{nym} \leftarrow \overline{cnym}_2 \overline{cnym}_1^{-bsk},\ m \leftarrow \overline{c}_2 \overline{c}_1^{-bsk}$
return $(\overline{nym}, m)$

4.2 Security of CLS–DDH

We now show that our scheme satisfies all security properties defined in Section 3. More precisely, we show that the following theorem holds (using the type-3 pairing version of the q-SDH assumption given in [8]).

Theorem 1. *The CLS–DDH construction presented in Sec. 4.1 is a secure CLS as defined in Sec. 3 if the DDH assumption holds in $\mathbb{G}_1$, the q-SDH assumption holds, and the SPK is simulation-sound, zero-knowledge and online extractable (for the underlined values).*

In the following we focus on the proof of non-transitivity which was the most challenging property to define and prove. For the other properties we provide short proof sketches and refer for the detailed proofs to the full paper [27].

Lemma 1. *The CLS–DDH construction presented in Sec. 4.1 satisfies **anonymity** if the DDH assumption holds in $\mathbb{G}_1$, and the SPK is unbounded simulation-sound, zero knowledge and online extractable (for the underlined values).*

Proof (sketch). Roughly, anonymity follows from the unlinkability property of BBS+ signatures, the CPA-security from ElGamal (used to compute the pseudonyms under cpk), and the DDH assumption (for showing that the conversion doesn't leak any information). Recall that in this setting, the converter is honest, i.e., $\mathcal{A}$ does not know csk but is given access to the CONVERT oracle. Thus, the surprising part might be that CPA encryption is sufficient, despite the converter having to decrypt the blinded pseudonyms. However, in the security proof we can simulate decryption queries by computing the converted pseudonyms from scratch and returning fresh encryptions of them (under bpk) to the adversary. That is, here we use that the convert algorithm returns *re-randomised* ciphertexts which, for ElGamal encryption, are distributed as fresh encryptions. To recover the plaintext, i.e., h^y that needs to be encrypted under bpk, we either look up h^y from our internal records (when the pseudonyms stem from honest users) or extract y from π (when the pseudonym belongs to a corrupt user). Thus, for each tuple (m_i, nym_i, σ_i) sent to the CONVERT we check if an entry $(uid_i, m_i, nym_i, \sigma_i)$ in the list of created signatures SL exist, and if so we look up the h^{y_i} value we have chosen when mimicking the join protocol for this honest user uid_i. For computing the converted pseudonyms, we then simply compute $\overline{cnym}_i = \mathsf{Enc}(bpk, h^{y_i r})$ for a fresh r. Note that in the case of pseudonyms from corrupt users it is not sufficient to extract just h^y, which would be much more efficient than extracting y: When we have to embed a DDH challenge in the converted output, we won't be privy of the converter's exponent r that is supposed to be used in all converted pseudonyms $h^{y_i r}$. Knowing y we can simply compute R^y for $R = g^r$ being a part of the DDH challenge.

□

Lemma 2. *The CLS–DDH construction presented in Sec. 4.1 satisfies **non-transitivity** if the DDH assumption holds in $\mathbb{G}_1$, and the SPK is unbounded simulation-sound, zero knowledge and online extractable (for the underlined value).*

Proof. For proving non-transitivity, we have to show that there exists an efficient simulator SIM that makes the real and simulated game indistinguishable. We start by describing the simulator and then explain why the real and simulated conversion oracles $\mathsf{CONVERT}$ and $\mathsf{CONVSIM}$ are indistinguishable.

$\mathsf{SIM}(gpk, bpk, \mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k'}})$

$l \leftarrow 0, \forall j \in [1, k']$
  $nym' \leftarrow_{\$} \mathbb{G}_1; \forall m \in L_{uid_j}$
    $l \leftarrow l + 1, (\overline{cnym}_l, \bar{c}_l) \leftarrow \mathsf{Blind}(gpk, bpk, (nym', m))$
return $((\overline{cnym}_1, \bar{c}_1), \ldots (\overline{cnym}_l, \bar{c}_l))$

We assume that an adversary $\mathcal{A}$ exists, that makes q queries to the SNDU oracle for distinct user identifiers, that guesses b correctly in the non-transitivity game with SIM given above and wins with probability $\epsilon + 1/2$.

We will stepwise make the real-world (b=0) and the simulated world (b=1) equivalent, using a sequence of Games $\mathcal{H}_j$ for $j = 0, \ldots, q$. The idea is that in Game $\mathcal{H}_j$ we will not use simulated conversions for all users $uid_1, \ldots, uid_j$ in order of when they were queried to SNDU. More precisely, we define Game $\mathcal{H}_j$ to be as given in Figure 3 with all other oracles identical to in the non–transitivity experiment. Let S_j be the event that $\mathcal{A}$ guesses b correctly in Game $\mathcal{H}_j$, with the simulator given above. Game $\mathcal{H}_j$ keeps track of the queries to SNDU, adding the first j queries uid to a set UL. Then during queries to $\mathsf{CONVSIM}$, if a signature of a user in UL is queried, these are treated in the same way as pseudonyms for corrupted users, i.e., they are normally converted using the $\mathsf{Convert}$ algorithm.

Game $\mathcal{H}_0$ is identical to the non-transitivity game, because UL is empty. Therefore, $\Pr[S_0] = \epsilon + 1/2$. In Game $\mathcal{H}_q$, UL contains all honest users, and so the $\mathsf{CONVSIM}$ oracle is now identical to the $\mathsf{CONVERT}$ oracle, and inputs to the adversary are now independent of b , therefore $\Pr[S_q] = 1/2$.

We now show that if an adversary can distinguish Games $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$, we can turn this into a distinguisher $\mathcal{D}_j$ that can break the DDH assumption. We describe the reduction and the additional simulation that is needed therein in Figures 4 and 5.

We now argue that when a DDH tuple (D_1, D_2, D_3, D_4) is input to $\mathcal{D}_j$, the inputs to $\mathcal{A}$ are distributed identically to in Game $\mathcal{H}_{j+1}$; when a DDH tuple is not input, the inputs to $\mathcal{A}$ are distributed identically to in Game $\mathcal{H}_j$. That is for $D_1 = h, D_2 = h^a, D_3 = h^b, D_4 = h^c$, the oracles provided by $\mathcal{D}_j$ will be exactly as in $\mathcal{H}_{j+1}$ when $c = ab$, and as in $\mathcal{H}_j$ otherwise.

First, note that gpk, csk, isk are distributed identically as to the non–transitivity game, as χ is chosen randomly and independently when setting $h_1 \leftarrow h^{\chi}$.

Simulating the SNDU Oracle The SNDU oracle only differs from the oracle in the non–transitivity experiment during the $(j + 1)$-th query by embedding D_2 of the DDH challenger into the user's "public key" H using knowledge of χ. Clearly, H is distributed identically as when computed normally, and π_H can be simulated due to the zero-knowledge property of the proof system. Note that y

Game $\mathcal{H}_j$

$t \leftarrow 0, b \leftarrow_{\$} \{0,1\}, \mathsf{param} \leftarrow \mathsf{Setup}(1^\tau)$
$\mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \varnothing$
$(ipk, isk) \leftarrow \mathsf{IKGen}(\mathsf{param}), (cpk, csk) \leftarrow \mathsf{CKGen}(\mathsf{param})$
$gpk \leftarrow (\mathsf{param}, ipk, cpk), b^* \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(\mathsf{guess}, gpk, isk)$
return b^*

$\mathsf{SNDU}(uid, M_{\mathsf{in}})$

if $uid \notin \mathsf{HUL}, t \leftarrow t+1,$ **if** $t \leqslant j \quad \mathsf{UL} \leftarrow \mathsf{UL} \cup \{uid\}$
Continue from line 5 of standard SNDU oracle

$\mathsf{CONVSIM}((nym_1, m_1, \sigma_1), \ldots, (nym_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1,k]$ s.t. $\mathsf{Verify}(gpk, m_i, nym_i, \sigma_i) = 0$ or $bpk \notin \mathcal{BPK}$ **return** $\perp$
Set $\mathsf{CL} \leftarrow \varnothing$
Compute $(cnym_i, c_i) \leftarrow \mathsf{Blind}(gpk, bpk, (nym_i, m_i); r_i)$ for $i = 1, \ldots, k$
$\forall i \in [1,k]$
  if $(uid, m_i, nym_i, \sigma_i) \in \mathsf{SL}$
    if L_{uid} does not exist, create $\mathsf{L}_{uid} \leftarrow \{m_i\}, \mathsf{CL}_{uid} \leftarrow \{(c_i, cnym_i)\}$
    else set $\mathsf{L}_{uid} \leftarrow \mathsf{L}_{uid} \cup \{m_i\}, \mathsf{CL}_{uid} \leftarrow \mathsf{CL}_{uid} \cup \{(c_i, cnym_i)\}$
  else $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(c_i, cnym_i)\}$
$\{(\overline{cnym}_i, \overline{c}_i)\}_{i=1,\ldots k'} \leftarrow \mathsf{Convert}(gpk, csk, bpk, \mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid})$
for $k' \leftarrow |\mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}|$
let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters created above
Let $\mathsf{NUL} \leftarrow \{uid_1, \ldots uid_{k''}\} \backslash \mathsf{UL}$
$\{(\overline{cnym}_i, \overline{c}_i)\}_{i=k'+1,\ldots k} \leftarrow \mathsf{SIM}(gpk, bpk, \bigcup_{uid \in \mathsf{NUL}} \mathsf{L}_{uid})$
Choose random permutation $\Pi; \forall i \in [1,k] \quad (\overline{cnym}'_i, \overline{c}'_i) \leftarrow (\overline{cnym}_{\Pi(i)}, \overline{c}_{\Pi(i)})$
return $(\{(cnym_i, c_i, r_i)\}_{i=1,\ldots,k}, \{(\overline{cnym}'_i, \overline{c}'_i)\}_{i=1,\ldots k}, r_1, \ldots, r_k)$

Fig. 3. Description of Game $\mathcal{H}_j$ and the changes to the SNDU and $\mathsf{CONVSIM}$ oracles.

$\mathcal{D}_j(D_1, D_2, D_3, D_4)$

$t \leftarrow 0, b, \leftarrow_\$ \{0,1\}, h \leftarrow D_1, \chi \leftarrow_\$ \mathbb{Z}_p^*, h_1 \leftarrow h^\chi$

Finish computing gpk, csk, isk as in $\mathsf{Setup}, \mathsf{IKGen}, \mathsf{CKGen}$

$\mathsf{HUL}, \mathsf{CUL}, \mathsf{SL} \leftarrow \varnothing$

$b^* \leftarrow \mathcal{A}^{\mathsf{SNDU},\mathsf{SIGN},\mathsf{CONVX}}(\mathsf{guess}, gpk, isk)$

return b^*

$\mathsf{SNDU}(uid, n)$

if $uid \notin \mathsf{HUL}, t \leftarrow t+1,$ **if** $t \leqslant j \quad \mathsf{UL} \leftarrow \mathsf{UL} \cup \{uid\}$

$\mathsf{HUL} \leftarrow \mathsf{HUL} \cup \{uid\}, \mathbf{gsk}[uid] \leftarrow \perp, M_{\mathsf{in}} \leftarrow \perp, \mathbf{dec}^{uid} \leftarrow \mathsf{cont}$

if $t = j+1 \quad uid' \leftarrow uid, H \leftarrow D_2^\chi$ simulate π_H with $H, n, st_{uid} \leftarrow (\perp, H, \pi_H)$

return $((H, \pi_H), \mathsf{cont})$

Continue from line 5 of standard SNDU oracle

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

if $uid \neq uid'$ perform SIGN oracle from Anonymity experiment

else $\alpha, \beta \leftarrow_\$ \mathbb{Z}_p^*, nym_1 \leftarrow g^\alpha, nym_2 \leftarrow cpk^\alpha D_2$

$A', d \leftarrow_\$ \mathbb{Z}_p^*, \hat{A} \leftarrow A'^{isk}$

Simulate π with $A', \hat{A}, d, nym_1, nym_2, m$

$\sigma \leftarrow (A', \hat{A}, d, \pi) \quad$ **return** $((nym_1, nym_2), \sigma)$

Fig. 4. Oracles for $\mathcal{D}_j$ our distinguishing algorithm for the DDH problem. The $\mathsf{CONVERT}$ oracle remains unchanged, and the $\mathsf{CONVSIM}$ oracle using the DDH challenge is given in Figure 4.

is not defined for this honest user, but this is not output to $\mathcal{A}$, or used in the next stage of the protocol.

Simulating the SIGN Oracle The SIGN oracle is identical to the oracle in the non–transitivity experiment, when $uid \neq uid'$ is queried. When uid' is queried, we simply encrypt D_2 instead of h^y.

This is consistent with SNDU, as $H = D_2^\chi$. Further, A', d' are chosen randomly and independently, and $\hat{A} = A'^{isk}$ and so these are distributed identically to in Sign. The SPK π can be simulated due to the zero knowledge property of the proof system.

Simulating the CONVSIM Oracle What remains to be shown is that the $\mathsf{CONVSIM}$ oracle created by $\mathcal{D}_j$ either behaves identical to the $\mathsf{CONVSIM}$ oracle in Game $\mathcal{H}_j$ or as in $\mathcal{H}_{j+1}$, depending on whether its input was a DDH tuple or not. We know that $D_3 = h^{\tilde{r}}$ for some $\tilde{r}$ and thus it must hold that $D_3^{yr} = h^{\tilde{r}ry}$. Finally, we derive $\overline{cnym}$ by encrypting D_3^{yr} from scratch under bpk, which is not noticeable to the adversary due to the re-randomisation that is applied in the conversion algorithm.

$\mathsf{CONVSIM}((nym_1, m_1, \sigma_1), \ldots, (nym_k, m_k, \sigma_k), bpk)$

if $\exists i \in [1,k]$ s.t. $\mathsf{Verify}(gpk, m_i, nym_i, \sigma_i) = 0$ or $bpk \notin \mathcal{BPK}$ **return** $\perp$

Set $\mathsf{CL} \leftarrow \varnothing, r \leftarrow_{\$} \mathbb{Z}_p^*$

Compute $(cnym_i, c_i) \leftarrow \mathsf{Blind}(gpk, bpk, (nym_i, m_i); r_i)$ for $i = 1, \ldots, k$

$\forall i \in [1,k]$

  if $(uid, m_i, nym_i, \sigma_i) \in \mathsf{SL}$

    if L_{uid} does not exist $\mathsf{L}_{uid} \leftarrow \{m_i\}, \mathsf{CL}_{uid} \leftarrow \{(m_i, y_{uid})\}$

    else $L_{uid} \leftarrow \mathsf{L}_{uid} \cup \{m_i\}, \mathsf{CL}_{uid} \leftarrow \mathsf{CL}_{uid} \cup \{(m_i, y_{uid})\}$

  else Extract y_i from $\sigma_i, \mathsf{CL} \leftarrow \mathsf{CL} \cup \{(m_i, y_i)\}$

let $\mathsf{L}_{uid_1}, \ldots \mathsf{L}_{uid_{k''}}$ be the non-empty message clusters created above

$n \leftarrow 0$; $\quad \forall (m,y) \in \mathsf{CL} \cup \bigcup_{uid \in \mathsf{UL}} \mathsf{CL}_{uid}$

  $n \leftarrow n+1, (\overline{cnym}_n, \overline{c}_n) \leftarrow \mathsf{Blind}(gpk, bpk, (D_3^{y r}, m))$

if $\mathsf{L}_{uid'}$ exists $\forall m \in L_{uid'}$

  $n \leftarrow n+1, (\overline{cnym}_n, \overline{c}_n) \leftarrow \mathsf{Blind}(gpk, bpk, (D_4^r, m))$

$\{(\overline{cnym}_i, \overline{c}_i)\}_{i=n+1,\ldots k} \leftarrow \mathsf{SIM}(gpk, bpk, \bigcup_{uid \in \mathsf{NUL}, uid \neq uid'} \mathsf{L}_{uid})$

Choose random permutation $\Pi; \forall i \in [1,k] \quad (\overline{cnym}'_i, \overline{c}'_i) \leftarrow (\overline{cnym}_{\Pi(i)}, \overline{c}_{\Pi(i)})$

return $(\{(cnym_i, c_i, r_i)\}_{i=1,\ldots,k}, \{(\overline{cnym}'_i, \overline{c}'_i)\}_{i=1,\ldots k}, r_1, \ldots, r_k)$

Fig. 5. The $\mathsf{CONVSIM}$ oracle used by distinguisher $\mathcal{D}_j$ given in Figure 5. To avoid confusion, we write uid' to refer to the $j+1$-th user that has joined the group (and for which $\mathcal{D}_j$ embedded the DDH challenge).

If (D_1, D_2, D_3, D_4) is a DDH tuple, then $D_4^r = h^{\tilde{r} r \tilde{y}}$. Therefore as $\tilde{y} = y_{uid'}$, the inputs to $\mathcal{A}$ are also distributed identically to in Game $\mathcal{H}_{j+1}$. Whereas if (D_1, D_2, D_3, D_4) is *not* a DDH tuple, then D_4^r, is distributed identically to nym', which was chosen randomly and independently. Therefore the inputs to $\mathcal{A}$ are distributed identically to in Game $\mathcal{H}_j$.

Reduction to the DDH problem Therefore the probability that $\mathcal{D}_j$ outputs 1 if it was given a valid DDH tuple as input is $\Pr[S_{j+1}]$, and $\Pr[S_j]$ is the probability that $\mathcal{D}_j$ outputs 1 when the input was not a DDH tuple. The advantage of $\mathcal{D}_j$ is then $|\Pr[S_j] - \Pr[S_{j+1}]|$, therefore $|\Pr[S_j] - \Pr[S_{j+1}]| = \epsilon_{\mathsf{DDH}}$.

Overall, for our sequence of games $\mathcal{H}_0$ to $\mathcal{H}_q$ it holds that $|\Pr[S_0] - \Pr[S_q]| \leqslant q\epsilon_{\mathsf{DDH}}$ and thus $\epsilon \leqslant q\epsilon_{\mathsf{DDH}}$ is negligible. This concludes our proof that the CLS–DDH construction satisfies non–transitivity. □

Lemma 3. *The CLS–DDH construction presented in Sec. 4.1 satisfies* ***conversion blindness*** *if the DDH assumption holds in* $\mathbb{G}_1$.

Proof (sketch). Given that all a corrupt converter sees are ElGamal ciphertexts that are encrypted under a key *bpk* for which *bsk* is not known to the adversary, the proof for conversion blindness is a straightforward reduction to the CPA-security of ElGamal which holds under the DDH assumption. □

Lemma 4. *The CLS–DDH construction presented in Sec. 4.1 satisfies* ***join anonymity*** *if the DDH assumption holds in $\mathbb{G}_1$, and the SPK is zero knowledge.*

Proof (sketch). For proving that adversary $\mathcal{A}$ cannot break the join anonymity of our CLS–DDH construction we have to show that it is infeasible to link a join session of an honest user to the user's signatures. In this setting the adversary controls both the converter and issuer. The only value the corrupt issuer learns during the join protocol from an honest user is $H = h_1^y$ for the user's secret y and π_H, the proof of knowledge of y. When receiving signatures from the user, the adversary can use the converter's secret key to recover h^y from nym and also sees π, the proof-of-knowledge of a BBS+ signature on y. By the zero-knowledge property of the proof system, neither π nor π_H leak any information about y. It is easy to see that an adversary that can link h_1^y and h^y for the independent generators h_1 and h can be turned into an adversary breaking the DDH assumption. □

Lemma 5. *The CLS–DDH construction presented in Sec. 4.1 satisfies* ***non-frameability*** *if the DL assumption holds in $\mathbb{G}_1$, and the SPK is simulation-sound and zero knowledge.*

Proof (sketch). If an adversary $\mathcal{A}$ exists that can break the non-frameability of our CLS–DDH scheme, then we can build an adversary $\mathcal{A}'$ that breaks the discrete logarithm assumption. Recall that non-frameability ensures that an adversary should not be able to create a valid signature that Convert will falsely link to signatures of an honest user. In the proof we embed re-randomized versions of a DL challenge $D = h^y$ in the join protocol for all users, i.e., using D^r instead of H when receiving the BBS+ signature from the corrupt issuer. We also set the public parameters such that $h_1 = h^z$ for a random exponent z. For signature queries we use the knowledge of z to compute proper looking pseudonyms, and then mimic the SPK by choosing A', d' randomly, setting $\hat{A} \leftarrow A'^{isk}$, and simulating π. If the adversary outputs his forgery (nym^*, σ^*, m) we extract y from π^* contained in σ^*. Clearly, this also relies on the simulation soundness and zero-knowledge property of the proof system. □

Lemma 6. *The CLS–DDH construction presented in Sec. 4.1 satisfies* ***traceability*** *if the q-SDH assumption holds, and the SPK is simulation-sound, zero knowledge and online extractable.*

Proof (sketch). We show that if an adversary $\mathcal{A}$ can break traceability for the CLS–DDH construction then we can build an adversary $\mathcal{A}'$ that breaks the q-SDH assumption. Roughly, to win the traceability game the adversary must be able to create more signatures that remain unlinkable in Convert than users he controls, which requires $\mathcal{A}$ to forge BBS+ signatures. Our proof closely follows the revised proof of the unforgeability of BBS+ signatures given in [14]. Note that this uses the newer version of the q-SDH assumption [8] that supports type-3 pairings, which in turn allows to prove the unforgeability of BBS+ signatures in the type-3 pairing setting. □

4.3 Instantiation of SPK and Efficiency

We now discuss how to securely instantiate the online-extractable SPK's used in our CLS–DDH construction and state the computational cost and lengths of signatures and pseudonyms.

Instantiation of SPK's. We have two non-interactive zero-knowledge proofs of knowledge in our scheme: π_H used in the join protocol for proving knowledge of y in $H = h_1^y$, and π proving knowledge of a BBS+ signature on y and that nym encrypts the same y. In both cases we need the witness y to be online extractable. For this, we additionally encrypt y under a public key that needs to be added to param (and to which in security proof we will know the secret key for), and extend π and π_H to prove that the additional encryption contains the same y that is used in the rest of the proof. For the verifiable encryption of y we use Paillier encryption [20], that is secure under the DCR assumption [36].

For transforming interactive into non-interactive zero-knowledge proofs we rely on the Fiat-Shamir heuristic that ensures security in the random oracle model. Due to this, we can now state Corollary 1.

Corollary 1. *The CLS–DDH construction presented in Sec. 4.1, with the SPK instantiated as above, is a secure CLS as defined in Sec. 3 under the DDH, q-SDH and DCR assumption in the random oracle model.*

Computational Cost. We give the operations required for the entities involved in the scheme in the table below. We denote k exponentiation in group $\mathbb{G}_i$ by $k\mathsf{exp}_{\mathbb{G}_i}$, k hash function calls by $k$hash, and k pairing operations by $k$pair. We denote k exponentiations in $\mathbb{Z}_{n^2}^*$ due to the Paillier encryption used, by $k\mathsf{exp}_{\mathbb{Z}_{n^2}^*}$.

Entity	Algorithm	Computational Cost
User	Sign	$16\mathsf{exp}_{\mathbb{G}_1} + 15\mathsf{exp}_{\mathbb{Z}_{n^2}} + 1\mathsf{hash}$
Verifier	Verify	$12\mathsf{exp}_{\mathbb{G}_1} + 11\mathsf{exp}_{\mathbb{Z}_{n^2}} + 1\mathsf{hash} + 2\mathsf{pair}$
	Blind	$6\mathsf{exp}_{\mathbb{G}_1}$
	Unblind	$2\mathsf{exp}_{\mathbb{G}_1}$
Converter	Convert(k pseudonyms input)	$7k\mathsf{exp}_{\mathbb{G}_1}$

Pseudonym & Signature Length. We give the sizes of pseudonyms and signatures in terms of the amount of group elements below. We denote the length required to represent k elements in $\mathbb{G}_1$ as $k\mathbb{G}_1$, k outputs of a hash function as kH, and k elements in $\mathbb{Z}_{n^2}^*$, due to the Paillier encryption used, as $k\mathbb{Z}_{n^2}^*$.

Pseudonym				Signature
Original nym	Blinded $cnym$	Converted $\overline{cnym}$	Unblinded Converted $\overline{nym}$	σ
$2\mathbb{G}_1$	$3\mathbb{G}_1$	$2\mathbb{G}_1$	$1\mathbb{G}_1$	$3\mathbb{G}_1$ $6\mathbb{Z}_p$ $1H$ $6\mathbb{Z}_{n^2}^*$

5 Conclusion and Future Work

In this work we have introduced a new form of group signatures that support flexible and controlled linkability: data can be collected in authenticated and fully unlinkable form, whilst still allow the data to be obliviously relinked by queries to a central entity. We have formalized the required security properties in a dynamic model, i.e., users are able to join the scheme, and proposed an efficient scheme that satisfies these requirement under discrete logarithm and Paillier assumptions in the random oracle model.

There are a number of open problem we consider to be interesting avenues for future work: Compared with the anonymity requirements of conventional *dynamic* group signatures, our anonymity notions are somewhat weaker as we do not allow the adversary to corrupt the two challenge users after it received the challenge signature. This means that our privacy related requirements do not yield *forward anonymity*. Given the conversion functionality that is inherent in our setting, achieving such stronger notion seems challenging, if not even impossible. In fact, for the related problem of group signatures with user-controlled linkability with signature-based revocation, forward anonymity has not been achieved by any of the existing schemes either.

Another direction for further work would be to investigate how to achieve security against fully malicious verifiers. On a high level, this will require to forward blinded versions of the users' signatures to the converter, allowing him to check the validity of the blinded inputs. The challenge is to do this while preserving the converter's capability to blindly decrypt and transform the inputs.

In a similar vein, we have considered the verifier to be both the data collector and data processor so far. However, our blind and unblind algorithms already cater for a more flexible setting, as they are specified in the public-key setting. That is, the verifier could blind and push the data to be linked towards a dedicated data processor holding the secret unblinding key. This has the advantage that data storage and processing can be strictly separated. For such a setting it might be desirable to preserve the authenticity of the data throughout the process, i.e., the blind conversion must also take the signatures as input and transform them into valid signatures for the re-linked pseudonyms.

Acknowledgments. The first author is supported by the UK Government as part of the CDT in Cyber Security program at Royal Holloway University of London (EP/K035584/1). The second author was supported by the European Union's Horizon 2020 research and innovation program under Grant Agreement Number 768953 (ICT4CART).

References

1. Eu general data protection regulation. `https://gdpr-info.eu`
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (Aug 2000)

3. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-TAA. In: Prisco, R.D., Yung, M. (eds.) SCN 06. LNCS, vol. 4116, pp. 111–125. Springer, Heidelberg (Sep 2006)
4. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (May 2003)
5. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (Feb 2005)
6. Bernhard, D., Fuchsbauer, G., Ghadafi, E., Smart, N.P., Warinschi, B.: Anonymous attestation with user-controlled linkability. International Journal of Information Security 12(3), 219–249 (2013)
7. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin and Camenisch [13], pp. 56–73
8. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. Journal of Cryptology 21(2), 149–177 (Apr 2008)
9. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (Aug 2004)
10. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J.: Foundations of fully dynamic group signatures. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16. LNCS, vol. 9696, pp. 117–136. Springer, Heidelberg (Jun 2016)
11. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Atluri, V., Pfitzmann, B., McDaniel, P. (eds.) ACM CCS 04. pp. 132–145. ACM Press (Oct 2004)
12. Brickell, E., Li, J.: A pairing-based daa scheme further reducing tpm resources. In: International Conference on Trust and Trustworthy Computing. pp. 181–195. Springer (2010)
13. Cachin, C., Camenisch, J. (eds.): EUROCRYPT 2004, LNCS, vol. 3027. Springer, Heidelberg (May 2004)
14. Camenisch, J., Drijvers, M., Lehmann, A.: Anonymous attestation using the strong diffie hellman assumption revisited. In: International Conference on Trust and Trustworthy Computing. pp. 1–20. Springer (2016)
15. Camenisch, J., Drijvers, M., Lehmann, A.: Universally composable direct anonymous attestation. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) PKC 2016, Part II. LNCS, vol. 9615, pp. 234–264. Springer, Heidelberg (Mar 2016)
16. Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized Schnorr proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (Apr 2009)
17. Camenisch, J., Lehmann, A.: (Un)linkable pseudonyms for governmental databases. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15. pp. 1467–1479. ACM Press (Oct 2015)
18. Camenisch, J., Lehmann, A.: Privacy-preserving user-auditable pseudonym systems. In: Security and Privacy (EuroS&P), 2017 IEEE European Symposium on. pp. 269–284. IEEE (2017)
19. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001)
20. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (Aug 2003)

21. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (Aug 1997)
22. Chaum, D.: Some weaknesses of "weaknesses of undeniable signatures" (rump session). In: Davies, D.W. (ed.) EUROCRYPT'91. LNCS, vol. 547, pp. 554–556. Springer, Heidelberg (Apr 1991)
23. ElGamal, T.: On computing logarithms over finite fields. In: Williams, H.C. (ed.) CRYPTO'85. LNCS, vol. 218, pp. 396–402. Springer, Heidelberg (Aug 1986)
24. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987)
25. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics 156(16), 3113–3121 (2008)
26. Galindo, D., Verheul, E.R.: Microdata sharing via pseudonymization. Joint UNECE/Eurostat work session on statistical data confidentiality (2007)
27. Garms, L., Lehmann, A.: Group signatures with selective linkability (2019), `https://eprint.iacr.org/2019/027`
28. Groth, J.: Fully anonymous group signatures without random oracles. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 164–180. Springer, Heidelberg (Dec 2007)
29. Hwang, J.Y., Lee, S., Chung, B.H., Cho, H.S., Nyang, D.: Short group signatures with controllable linkability. In: Lightweight Security & Privacy: Devices, Protocols and Applications (LightSec), 2011 Workshop on. pp. 44–52. IEEE (2011)
30. Hwang, J.Y., Lee, S., Chung, B.H., Cho, H.S., Nyang, D.: Group signatures with controllable linkability for dynamic membership. Information Sciences 222, 761–778 (2013)
31. Kiayias, A., Tsiounis, Y., Yung, M.: Traceable signatures. In: Cachin and Camenisch [13], pp. 571–589
32. Kiayias, A., Yung, M.: Group signatures with efficient concurrent join. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 198–214. Springer, Heidelberg (May 2005)
33. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 1–31. Springer, Heidelberg (May 2016)
34. Libert, B., Peters, T., Yung, M.: Scalable group signatures with revocation. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 609–627. Springer, Heidelberg (Apr 2012)
35. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H.M., Adams, C.M. (eds.) SAC 1999. LNCS, vol. 1758, pp. 184–199. Springer, Heidelberg (Aug 1999)
36. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (May 1999)
37. Slamanig, D., Spreitzer, R., Unterluggauer, T.: Adding controllable linkability to pairing-based group signatures for free. In: International Conference on Information Security. pp. 388–400. Springer (2014)