Nicknames for Group Signatures: auditable transfers

Guillaume Quispe[1], Pierre Jouvelot[1,2], and Gérard Memmi[1]

[1] LTCI, Télécom Paris, IP Paris, Palaiseau, France
[2] Mines Paris, PSL University, Paris, France
`firstname.lastname@telecom-paris.fr`

Abstract. In this paper, we introduce Nicknames for Group Signatures (NGS), a new signature scheme for auditable transfers that builds upon group signatures and flexible public keys. With group signatures, a signer can prove membership of a group controlled by a manager, while only an opener is able to identify him. Meanwhile, signatures with flexible public keys allow anyone to derive different public keys for a particular user, who can detect them and sign with them; in a multi-user setting, this scheme allows a direct application: stealth addresses. With NGS, group members now expose their flexible public keys, named "master public keys", enabling anonymous yet auditable transfers. We describe NGS along with its security model and provide a mathematical implementation that we prove secure in the Random Oracle Model (ROM). To illustrate its practicality, we build NickHat, a blockchain-based token-exchange prototype system on top of Ethereum.

Keywords: Auditability · Group signature · Nicknames · Privacy

1 Introduction

Privacy is a fundamental property that safeguards users' information and communications. It is also enforced by legal frameworks such as the General Data Protection Regulation [16]. A variety of cryptographic schemes, or "primitives", aim to provide different solutions to enable this privacy. For instance, blind signatures [10] conceal the message being signed, while ring signatures [29] hide the signer's identity within the ring he creates. Signatures with flexible public keys (SFPK)[2] allow anyone to designate a user anonymously by creating a new identity for him.

But full anonymity may not always be desirable as it can protect abuses and malicious behaviors. In many scenarios, some control is needed. To this end, primitives such as traceable ring signature [15] sanction users when they misbehave, e.g. when they vote more than once, by revealing their identity; in other words, cheaters will be exposed. Group signatures (GS) [11] offer a stronger control by introducing an opener, the only entity capable of identifying any member of the group by its signature; otherwise, the signature only reveals his belonging to the group.

Following GS, Kiayias et al [23] adds control on encryption and introduces an interesting primitive called Group Encryption (GE) that hides the identity of a ciphertext recipient while still guaranteeing his belonging to the group. Additionally, an opening authority can reveal the identity of this recipient, i.e., the member capable of decrypting the message. Instead of encryption, our paper focuses on identity and ownership by introducing control over flexible public keys, i.e., SFPK, thus enabling auditable transfers.

Nicknames for Group Signatures We propose a new primitive called "Nicknames for Group Signatures" (NGS) that lets anyone generate a new identity, called "nickname", for a group member, while still ensuring group membership of this new identity. The targeted member can then retrieve and prove ownership of his nickname with signing capabilities. This new identity cannot be linked to the targeted group member, except by the latter and an opening authority. This proposal thus merges concepts from GS and SFPK and can be thought as an extension of group signatures.

On practical grounds, we believe the NGS primitive to be particularly relevant in scenarios where users or entities must be registered and their interactions or connections concealed from the public, while yet remaining auditable. Such activities include auditable payment systems [28], supply-chain management [12], crowdfunding [31] and car sharing systems [30]. More generally, we believe that NGS offers a generic solution that can be used there, thus avoiding domain-specific approaches. Indeed, NGS has already been put to practical use into two such different applications. The first one is NickPay [28], an NGS-based payment system built on top of the very popular Ethereum blockchain [8]. The second is NickHat, a blockchain-based token-exchange system (see Appendix D) that supports auditing and provides anonymity.

Implementation We implement NGS with a focus on efficiency, since we are currently targeting practical financial applications on Ethereum (see Appendix D). Therefore, instead of SPS-EQ-based [19] GS, the design of NGS relies here on GS built upon randomizable signatures [6] [27]. We prove the construction secure in Random Oracle Model (ROM) for the NGS security model we propose.

Contributions In this paper, we introduce the following contributions:

– the formal definition of the Nicknames for Group Signatures scheme (NGS);
– the security properties (correctness, non-frameability for tracing and opening, traceability, optimal opening soundness, opening coherence, and anonymity) that NFS enforces;
– a mathematical implementation of NGS, based on [24], with a thorough analysis of NGS security properties;
– the NickHat blockchain application, based on a new Rust computer implementation of NGS (not described here, for lack of space).

Organization After this introduction, Section 2 covers the related work. In Section 3, we introduce NGS along with its security model. We then describe, in Section 4, an efficient NGS implementation based on [24], together with its associated security proofs (these are available as `XXXXX`). Finally, Section 5 concludes and offers some perspectives for possible future work.

2 Related work

If NGS is based on traditional group signatures, we highlight here its key differences (see below for the functions' definitions). First, the *Open* and *Judge* functions only take as argument nicknames, not signatures. Second, NGS introduces the *Nick* and *Trace* functions, enabling users to track the nicknames in their equivalence class. Finally, the verification function is here split in two functions, *GVf* and *UVf*, while a register of master public keys is published.

Group signature systems that associate members to equivalence classes have been proposed previously [13][2]. Equivalence classes are used in the definition of NGS, enabling SFPK integration and better understanding. Compared to SFPK [2], NGS adds the opening and group management features. SFPK's property of class hiding is strictly stronger than NGS's anonymity, as a NGS adversary is not given the secret key of the target user [7]; their unforgeability property is, however, encompassed by NGS's non-frameability and traceability properties (see Section 3.4).

When introducing SFPK, the authors additionally proposed a group-signature construction based on structure-preserving signatures with equivalence classes (SPS-EQ) and SFPK. During the joining protocol, the issuer signs, with SPS-EQ, a user's public key representation of SFPK. In order to sign a message, the member first randomizes his SFPK public key, then has to adapt the SPS-EQ representation with the same randomizer. Therefore, this requires, in the end, not only more computation than NGS' proof-of-knowledge-based approach but also three-times-larger signatures than the ROM ones. Our construction of NGS, based on [24], builds upon an efficient proof-of-knowledge approach as it takes place in $\mathbb{G}_1$ (see Section 4), matching the performance required by our targeted applications (see, e.g., Appendix D). It thus avoids the two randomization operations during the signature process, while preserving the efficiency of proof-of-knowledge techniques.

In [1], the authors propose a public-key anonymous tag system, where users can create tags with their private key, thus for themselves, and the authority can publish a trapdoor (or tracing token) that publicly link all tags related to one user without revealing who it is. But the same trapdoor can be computed by this user, enabling him to then prove or deny the link between a tag and his public key. As in NGS, the authority and the user share the same trapdoor, but no means to contact users is provided.

With traceable signatures [22], the concern was about protecting honest users' anonymity while still enabling the identification of the malicious ones. A trusted party can ask the group manager to investigate on suspect signatures, and the latter returns a trapdoor of the suspect, allowing the trusted party to open all signatures of the suspect members while the anonymity of the other members is protected. They also defined the self-traceability property, where a member is able to claim that a signature is his. It can be seen as group signatures with additional anonymity management, with no method to specify a member.

With pseudonym systems [26] and anonymous credentials, users ask organizations for a credential on attribute(s) so that they can claim possession of it anonymously. To register with an organization, usually the user and the organization together compute a pseudonym for the former. In [9], the first anonymous credential with optional anonymity revocation is proposed. In [20], the authors build an anonymous-credentials system based on two primitives they introduce. With the first one, which concerns us here, named *EphemerId*, users are associated with a tag pair, a secret and its corresponding public tags, which they can randomize and derive the witness for. They also extend this primitive with a tracing authority that can identify users given any public randomized tag. Note that they, however, keep the general idea behind anonymous credentials, thus not offering a private and auditable transfer system.

3 Nicknames for Group Signatures

We describe Nicknames for Group Signatures (NGS), a new digital signature scheme that adds the concept of "nicknames" on top of the notion of group signatures. Let l denote the dimension of the space ${\mathbb{G}^*}^l$ of public keys pk, represented as tuples, where $\mathbb{G}$ is some cyclic group of some prime order p. We introduce an equivalence relation $\mathcal{R}$ that partitions ${\mathbb{G}^*}^l$ into equivalence classes: $\mathcal{R} = \{(pk_1, pk_2) \in ({\mathbb{G}^*}^l)^2 \mid \exists s \in \mathbb{Z}_p^*, pk_1 = {pk_2}^s\}$, where exponentiation on tuples is defined element-wise.

During the joining phase of the NGS protocol by user i, the issuer associates an equivalence class $[mpk]_\mathcal{R}$ to i and publishes the user's master public key mpk, an element of ${\mathbb{G}^*}^l$. NGS then allows anyone to transform mpk into a "nickname", i.e., a different representative nk of the same class $[mpk]_\mathcal{R}$, without accessing the secret key controlling mpk. To ensure anonymity, two nicknames of the same equivalence class cannot be linked. However, a user i can retrieve all the nicknames in his class $[mpk]_\mathcal{R}$ thanks to his NGS-provided trapdoor, while the opener can identify the member hidden behind a nickname, by iterating over all the users' trapdoors, and even prove the validity of his results to a judge.

3.1 Interface

NGS is abstracted over some roles, types, variables, and functions that will be instantiated when defining a precise implementation (see Section 4).

Definition 1 (NGS Roles). *In the NGS scheme, each participant has a role that provides him with rights to handle particular data or variables and abilities to perform parts of the NGS scheme. We describe the five key roles of our scheme.*

- *A User becomes a group member by having his join request accepted by the issuer. Then, he will be able to produce nicknames and sign messages.*
- *The Issuer authorizes users to become group members.*
- *The Opener is the only participant able to "open" a nickname to unveil the underlying group member's identity, together with a proof of it.*
- *Anyone can act as a Verifier, to check that a given nickname does exist.*
- *The Judge role can be adopted to check that a proof supposedly linking a user to a nickname (following the opening of a nickname) is valid.*

One could also define the additional role of Group Manager, which would endorse the roles of issuer and opener. At last, the Adversary can also be considered as a role in his own right and will be defined in Section 3.3.

Definition 2 (NGS Types). *NGS uses the following set of generic types.*

- *A Registration information is a structure, usually named reg, that contains the necessary elements of a user, saved in the registration table* **reg** *defined below. The exact contents of reg is implementation-dependent, but can include, for example, the encryption of the users' trapdoors τ.*
- *A Join request is a structure, named $reqU$, produced by a user requesting to join the group. $reqU$ contains the necessary implementation-dependent elements for a joining request to be handled by the issuer. It can be seen as a synchronization element between the user and the issuer during the group-joining algorithm.*

Definition 3 (NGS Global Variables). *NGS is built on the following set of global variables.*

- *$DS = \{KeyGen, Sig, Vf\}$ is a digital signature scheme vetted by a certification authority CA.*
- **reg** *is the registration table controlled by the issuer who has read and write access to it. The opener is given a read access to it also.*
- **mpk** *is the master public key table, publicly available, and used to define nicknames.*
- **upk** *is the publicly available table of users' public keys.*

Definition 4 (NGS Scheme). *A* nicknames for group signature scheme *(NGS) is a tuple of functions, each one particularly related to one key role in NGS-based protocols, (User, Issuer, Opener, Verifier, Judge), defined as follows.*

- *$IKg : 1^\lambda \to (isk, ipk)$ is the key generation algorithm that takes a security parameter λ and outputs an issuer's secret/public keys (isk, ipk).*
- *$OKg : 1^\lambda \to (osk, opk)$ is the key generation algorithm that takes a security parameter λ and outputs an opener's secret/public keys (osk, opk).*
- *$UKg : 1^\lambda \to (usk, upk)$ is the user key generation algorithm that produces appropriate DS secret/public keys.*

- *Join* : $(usk, ipk, opk) \to (msk, \tau, reqU)$*, the user part of the group-joining algorithm, takes a user's secret key* usk *and the issuer and opener public keys and outputs a master secret key* msk *along with a trapdoor* τ*.* $reqU$ *will then contain information necessary to the Issuer to run the second part of the group-joining algorithm.*
- *Iss* : $(i, isk, reqU, opk) \to ()$ *or* $\perp$*, the issuer part of the group-joining algorithm, takes a user* i*, an issuer secret key* isk*, a join request* $reqU$ *and the opener public key* opk*, updates the registration information* $\mathbf{reg}[i]$ *and master public key* $\mathbf{mpk}[i]$*, and creates an equivalence class for the user nicknames. He returns* $\perp$*, in case of unsuccessful registration.*
- *Nick* : $(mpk, r) \to nk$ *is the nickname generation algorithm that takes a master public key* mpk*, a random coin* r*, and creates a nickname* nk *belonging to* $[mpk]_{\mathcal{R}}$*.*
- *Trace* : $(ipk, \tau, nk) \to b$ *takes as input the issuer public key* ipk*, the trapdoor* τ *for some equivalence class* $[mpk]_{\mathcal{R}}$*, and a nickname* nk *and outputs the boolean* $b = (nk \in [mpk]_{\mathcal{R}})$*.*
- *Sig* : $(nk, msk, m) \to \sigma$ *takes a nickname* nk*, a master secret key* msk *and the message* m *to sign and outputs the signature* σ*.*
- *GVf* : $(ipk, nk) \to b$ *is the issuer verification algorithm that takes a issuer public key* ipk *and a nickname* nk *and outputs a boolean stating whether* nk *corresponds to a user member of the group or not.*
- *UVf* : $(nk, m, \sigma) \to b$ *is the user verification algorithm, taking as input a nickname* nk*, a message* m *and a signature* σ *of* m *and outputting a boolean stating whether* σ *is valid for* m *and* nk *or not.*
- *Open* : $(osk, nk) \to (i, \Pi)$ *or* $\perp$*, the opening algorithm, takes an opener secret key* osk *and a nickname* nk*. It outputs the index* i *of a user with a proof* Π *claiming that user* i *controls* nk *or* $\perp$*, if no user has been found.*
- *Judge* : $(nk, ipk, i, \Pi) \to b$ *is the judging algorithm that takes a nickname* nk*, an issuer public key* ipk*, a user id* i *and an opener's proof* Π *to be verified and, using the user public key from the CA* $\mathbf{upk}[i]$ *for user* i*, outputs a boolean stating whether the judge accepts the proof or not.*

3.2 Protocol

We describe here the main protocol steps of a typical application that uses NGS to ensure the security properties that NGS provides (see Section 3.4).

Setup The issuer and opener run IKg and OKg, respectively, to obtain their pair of secret/public keys (isk, ipk) and (osk, opk) respectively . The issuer will decide (or not) to grant users access to the group it manages, thus enabling the opener to track, i.e., open, their nicknames used in messages' signatures.

Group-joining synchronization process Prior to joining the group, the user i must run UKg to obtain his secret/public keys (usk, upk) and store upk in $\mathbf{upk}[i]$. Then, to enable the user i to join the issuer's group, the following three steps must be performed.

1. The user i must run the $Join$ function, providing notably a join request and his secret group signing key msk.
2. Meanwhile, the issuer must run the Iss function to verify the correctness of this join request.The issuer creates the user i's master public key, mpk, used to create his equivalence class and associated nicknames, and adds it to the public **mpk** table.
3. Finally, after the user checked that the issuer signature is correct, i.e., $GVf(ipk, mpk)$, he can privately store msk.

If the protocol succeeds, user i obtains its master secret key msk for the equivalence class $[mpk]_{\mathcal{R}}$, allowing him to sign on behalf of the group, along with his trapdoor τ.

Nickname creation Anyone can now create a nickname for user i by calling the $Nick$ function on his master public key $\mathbf{mpk}[i]$. It returns a nickname nk that only user i can control, i.e., prove possession of.

Group verification Any verifier can check if the nickname nk is part of the group by running GVf with the issuer public key ipk.

User tracing User i can check if he can unlock a particular nickname nk, meaning that nk is in the equivalence class $[\mathbf{mpk}[i]]_{\mathcal{R}}$, by calling the $Trace$ function with its trapdoor τ. This function can be used as a subroutine of the signing protocol.

Signing After tracing, the user i knows he controls nk. He can then produce a signature σ on a message m with his master secret key msk.

User Verification Any verifier can check that the signer controls nk by verifying the signature σ.

Opening The opener can identify which user controls a nickname nk. When successful, he produces a publicly verifiable proof Π stating that user i indeed controls nk.

Judging A judge can verify, at any time, such a proof Π via the public $Judge$ algorithm, thus being assured that user i indeed controls nk.

3.3 Security model

As stated in the introduction, GS has been studied for several decades and formalized [4][3]. It therefore constitutes a solid basis for NGS to build its security model on [4], with the necessary modifications that we highlight here.

The NGS traceability property, similar to the GS one, states that any "NGS-signed nickname", i.e., any triplet (m, nk, σ) passing the GVf and UVf functions can be opened by the NGS $Open$ function, i.e., the member who signed it can be identified.

The NGS non-frameability property protects honest users from being falsely accused of producing a signature σ, while also protecting users from tracing a nickname that was not signed by them. The experiment requires $\mathcal{A}$ to output a forgery consisting of a nickname nk satisfying GVf, along with its corresponding signature σ satisfying UVf on nk for a given message m. This game is won when a proof Π convinces the Judge that it opens to an honest user controlled by the challenger $\mathcal{C}$ or when an honest user traces the valid nickname.

The NGS anonymity experiment considers IND-CPA anonymity. $\mathcal{A}$ is restricted, with no access to a *Trace* or *Open* oracle, and the *Sig* oracle that implicitly traces the nickname is replaced with a *SigR* oracle.

Requirement	Opener	Issuer
Anonymity	honest	fully corrupt
Traceability	partially corrupt	honest
Non-frameability	fully corrupt	fully corrupt
Opening soundness	fully corrupt	honest
Opening coherence	fully corrupt	honest

Table 1. Requirements for the opener and issuer in NGS security model

Below, we present the oracles used in our security model. Then we present, in the subsequent subsection, the NGS correctness, traceability, non-frameability, optimal opening soundness, opening coherence and anonymity properties (see Table 1).

Oracle description In this section, we introduce the oracles used in our security experiments (see Figure 1). We assume an NGS-based system for one group, with its issuer and opener, attacked by an adversary $\mathcal{A}$ able to take advantage of some of these oracles, plus a *Hash* function that embeds the Random Oracle Model.

- AddU(i): $\mathcal{A}$ uses this oracle to add and then join an honest user i.
- CrptU(i, upk): $\mathcal{A}$ uses this oracle to corrupt user i and set its public key to be a upk of its choice.
- SndToI(i, r): $\mathcal{A}$ uses it to send the join request r of a malicious user i to an honest issuer executing *Iss*. $\mathcal{A}$ does not need to follow the *Join* algorithm.
- SndToU(i): $\mathcal{A}$ uses this oracle to accept or not an honest user i. $\mathcal{A}$ does not need to follow the *Iss* algorithm.
- USK(i): $\mathcal{A}$ uses this oracle to get the secret keys $\mathbf{msk}[i]$ and $\mathbf{usk}[i]$ of an honest user i.
- RReg(i): $\mathcal{A}$ can read the entry i in the registration table **reg** for user i.
- WReg(i, ρ): with this oracle, $\mathcal{A}$ can write or modify the entry for user i in the registration table **reg**.
- Sig(i, nk, m): $\mathcal{A}$ uses this oracle to obtain a signature on a message m from user i on nickname nk.
- SigR(i, r, m): this oracle restricts $\mathcal{A}$ to use Sig on a nickname honestly generated with the random coin r provided.
- Ch$_b$(i_0, i_1, m): for two users i_0 and i_1 and a message m chosen by $\mathcal{A}$, this oracle outputs a NGS-signed nickname (m, nk, σ) on m under identity i_b as the challenge.

Below is the set of lists maintained by the challenger $\mathcal{C}$ to control the oracles (all are initially empty):

- L_h: list of honest users with their trapdoors;
- L_c: list of corrupted users and their current state, *cont* or *accept* (*cont* indicates that the user is corrupted but not yet joined, and *accept* indicates that the user is corrupted and also accepted to join the system by the issuer);
- L_{ch}: list of challenged messages and identities, along with a signature, in response to the challenge oracle;
- L_{sk}: list of user identities queried by $\mathcal{A}$ to access their secret keys via *USK*.
- L_{nk}: list of nicknames queried by the adversary;
- L_σ: list of queried identities, messages, nicknames and signatures in response to the signing oracle.

Finally, we assume that three additional data structures are added to the NGS implementation, with no change otherwise to its semantics. The tables **usk** (updated in UKg) and **reqU** and **msk** (updated in $Join$) keep the user secret keys, the user requests and the master secret keys; they are only available within the oracles and challenger.

3.4 Security properties

Correctness Let m be any message that a user i, after joining the group, signs with his secret key msk while being nicknamed nk. Correctness ensures that the opener cannot open m to a user i' different from i. Also, the verification functions validate nk and (m, σ), while the proof Π generated by *Open* must be accepted by the $Judge$ algorithm and nk traceable to i, given his trapdoor τ.

Definition 5. *An NGS is* correct *iff* $\Pr[Exp^{Corr}_{NGS}(\lambda, i, m)] = 1$ *for any parameter* λ*, user* i *and message* m *(see Figure 2).*

Traceability Traceability ensures that no adversary can create a valid NGS-signed nickname opening to a non-registered user, i.e., untraceable. In this experiment, the opener is partially corrupted, meaning that the opening must follow the prescribed program but the adversary $\mathcal{A}$ is given the opening key osk. The issuer is kept honest. $\mathcal{A}$ can call *AddU* to join an honest user, *CrptU* and *SndToI*, to join a corrupt user, and the *USK* oracle, to get all users secret keys (the *Sig* oracle is therefore not needed). Finally, $\mathcal{A}$ can read the registration list with *RReg*.

The adversary wins if he produces a valid NGS-signed nickname from a user who did not join the group, causing the *Open* algorithm to fail to identify the user[3]. He can also win if a forgery σ produced by some user i leads to a proof Π from the *Open* algorithm that gets rejected by the $Judge$ algorithm.

We define the advantage in the traceability experiment described in Figure 3 for any polynomial time adversary

$$\mathcal{A} = \mathcal{A}^{AddU, CrptU, SndToI, USK, RReg}$$

[3] Note that, since the operations performed in *Trace* are a subset of those of *Open*, this experiment also ensures that an invalid tracing cannot be forged.

```
AddU(i)
    $(usk, upk) = UKg(1^\lambda); \mathbf{upk}[i] = upk$
    $(msk, \tau, reqU) = Join(usk, opk)$
    $Iss(i, isk, reqU, opk)$
    $L_h = L_h \cup \{(i, \tau)\}$
    return $upk$

SndToI(i, r)
    If $(i, cont) \notin L_c$, return $\perp$
    $L_c = L_c \setminus \{(i, cont)\} \cup \{(i, accept)\}$
    return $Iss(i, isk, r, opk)$

SndToU(i)
    $(usk, upk) = UKg(1^\lambda); \mathbf{upk}[i] = upk$
    $(msk, \tau, reqU) = Join(usk, opk)$
    $L_h = L_h \cup \{(i, \tau)\}$
    return $reqU$

$Ch_b(i_0, i_1, m)$
    If $(i_0, \tau) \notin L_h \vee (i_1, \tau) \notin L_h$ for some $\tau$,
      return $\perp$
    If $i_0 \in L_{sk} \vee i_1 \in L_{sk}$, return $\perp$
    If $\mathbf{mpk}[i_b]$ undefined, return $\perp$
    $r \leftarrow_\$ \mathbb{Z}_p; nk = Nick(\mathbf{mpk}[i_b], r)$
    $\sigma = Sign(nk, \mathbf{msk}[i_b], m)$
    $L_{ch} = \{(i_0, m, nk, \sigma), (i_1, m, nk, \sigma)\} \cup L_{ch}$
    return $(nk, \sigma)$

USK(i)
    If $(i, m, nk, \sigma) \in L_{ch}$ for some $(m, nk, \sigma)$,
      return $\perp$
    $L_{sk} = L_{sk} \cup \{i\}$
    return $(\mathbf{msk}[i], \mathbf{usk}[i])$

RReg(i)
    return $\mathbf{reg}[i]$

WReg(i, ρ)
    $\mathbf{reg}[i] = \rho$

CrptU(i, upk)
    $\mathbf{upk}[i] = upk$
    $L_c = L_c \cup \{(i, cont)\}$

Sig(i, nk, m)
    If $(i, \tau) \notin L_h$ for some $\tau$, return $\perp$
    If $\neg Trace(ipk, \tau, nk)$ return $\perp$
    $\sigma = Sign(nk, \mathbf{msk}[i], m)$
    $L_\sigma = L_\sigma \cup \{(i, m, nk, \sigma)\}$
    return $\sigma$

SigR(i, r, m)
    If $(i, *) \notin L_h$, return $\perp$
    $nk = Nick(\mathbf{mpk}[i], r)$
    $\sigma = Sign(nk, \mathbf{msk}[i], m)$
    $L_\sigma = L_\sigma \cup \{(i, m, nk, \sigma)\}$
    return $\sigma$
```

Fig. 1. Oracles for the security model of NGS ($\leftarrow_\$ S$ denotes the uniform sampling function in the finite set S)

```
$Exp^{Corr}_{NGS}(\lambda, i, m)$
    $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
    $(usk, upk) = UKg(1^\lambda); \mathbf{upk}[i] = upk$
    $(msk, \tau, reqU) = Join(usk, opk)$
    $Iss(i, isk, reqU, opk)$
    $r \leftarrow_\$ \mathbb{Z}_p; nk = Nick(\mathbf{mpk}[i], r)$
    $\sigma = Sign(nk, msk, m)$
    $(i', \Pi) = Open(osk, nk)$
    return $\begin{pmatrix} (i = i') \wedge \\ GVf(ipk, nk) \wedge UVf(nk, m, \sigma) \wedge \\ Judge(nk, i, ipk, \Pi) \wedge Trace(ipk, \tau, nk) \end{pmatrix}$
```

Fig. 2. Correctness experiment for NGS

as $Adv^{Trace}_{NGS,\mathcal{A}}(\lambda) = \Pr[Exp^{Trace}_{NGS,\mathcal{A}}(\lambda) = 1]$.

Definition 6. *A NGS scheme is* traceable *if $Adv^{Trace}_{NGS,\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and λ.*

Non-frameability Non-frameability ensures two properties. Similar to GS, the first one protects users from being falsely accused of having signed a message. The second one prevents any (PPT) adversary from producing a valid signature for a valid nickname that an honest user can trace. The issuer and opener are both controlled by $\mathcal{A}$, so he receives osk and isk. He can use the *SndToU* oracle to add a new honest user to the group (the *CrptU*, *WReg* and *Open* oracles are therefore not needed). $\mathcal{A}$ has also access to the *USK* and *Sig* oracles.

The goal of such an adversary $\mathcal{A} = \mathcal{A}^{SndToU, USK, Sig}$ is to produce a forgery (nk^*, σ^*) on a message m^* passing the *GVf* and *UVf* verifications, such that either an honest user trace nk^*, either the target identity i^* and proof Π^* additionally provided are accepted by the *Judge* algorithm.

We define the advantage in the non-frameability experiment described in Figure 3 for any polynomial time adversary $\mathcal{A}$ as:

$$Adv^{Nf}_{NGS,\mathcal{A}}(\lambda) = \Pr[Exp^{Nf}_{NGS,\mathcal{A}}(\lambda) = 1].$$

Definition 7. *A NGS scheme is* non-frameable *if $Adv^{Nf}_{NGS,\mathcal{A}}(\lambda)$ is negligible, for any $\mathcal{A} = \mathcal{A}^{SndToU, USK, Sig}$ and any λ.*

Optimal opening soundness Optimal opening soundness guarantees that no adversary can produce a group-belonging nickname that can be opened to two

```
$Exp^{Trace}_{NGS,\mathcal{A}}(\lambda)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $(m^*, nk^*, \sigma^*) = \mathcal{A}^{AddU, CrptU, SndToI, USK, RReg}(ipk, osk)$
  If following conditions hold, then return 1:
    – $GVf(ipk, nk^*) \wedge UVf(nk^*, m^*, \sigma^*)$;
    – Let $r = Open(osk, nk^*)$ in
        $r = \perp \vee \neg Judge(nk^*, i^*, ipk, \Pi)$, when $(i^*, \Pi) = r$.
  return 0

$Exp^{Nf}_{NGS,\mathcal{A}}(\lambda)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $(m^*, nk^*, \sigma^*, i^*, \Pi^*) = \mathcal{A}^{SndToU, USK, Sig}(isk, osk)$
  If the following conditions hold, for some $\tau^*$, then return 1:
    – $GVf(ipk, nk^*) \wedge UVf(nk^*, m^*, \sigma^*)$;
    – $(i^*, \tau^*) \in L_h \wedge i^* \notin L_{sk} \wedge (i^*, m^*, nk^*, *) \notin L_\sigma$;
    – $Judge(nk^*, i^*, ipk, \Pi^*) \vee Trace(ipk, \tau^*, nk^*)$.
  return 0

$Exp^{OS}_{NGS,\mathcal{A}}(\lambda)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $(nk^*, i^*_0, \Pi^*_0, i^*_1, \Pi_1) = \mathcal{A}^{AddU, CrptU, SndToI, USK, RReg}(ipk, osk)$
  If the following conditions hold, then return 1:
    – $GVf(ipk, nk^*)$;
    – $Judge(nk^*, i^*_0, ipk, \Pi^*_0) \wedge Judge(nk^*, i^*_1, ipk, \Pi^*_1)$;
    – $i^*_0 \neq i^*_1$.
  return 0

$Exp^{OC}_{NGS,\mathcal{A}}(\lambda)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $(nk^*, i^*, \Pi^*) = \mathcal{A}^{AddU, CrptU, SndToI, USK, RReg}(ipk, osk)$
  If the following conditions hold, for some $(i, \tau)$, then return 1:
    – $GVf(ipk, nk^*)$;
    – $(i, \tau) \in L_h \wedge i \neq i^*$;
    – $Trace(ipk, \tau, nk^*) \wedge Judge(nk^*, i^*, ipk, \Pi^*)$.
  return 0

$Exp^{Anon-b}_{NGS,\mathcal{A}}(\lambda)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $b' = \mathcal{A}^{SndToU, USK, WReg, SigR, Ch_b}(opk, isk)$
  return $b'$
```

Fig. 3. Traceability, Non-frameability, Optimal opening soundness, Opening coherence and Anonymity (from top to bottom) experiments for NGS

distinct users. In this experiment, $\mathcal{A}$ can corrupt all entities, including the opener and all users, but not the issuer. He can use the *AddU*, *CrptU*, *SndToI*, *USK* and *RReg* oracles. The adversary wins if he produces a nickname nk with two opening users and proofs (i_0, Π_0) and (i_1, Π_1), where $i_0 \neq i_1$, both accepted by the $Judge$ algorithm[4]. We define the advantage in the optimal opening soundness experiment described in Figure 3 for any polynomial time adversary $\mathcal{A} = \mathcal{A}^{AddU, CrptU, SndToI, USK, RReg}$ as

$$Adv^{OS}_{NGS,\mathcal{A}}(\lambda) = \Pr[Exp^{OS}_{NGS,\mathcal{A}}(\lambda) = 1].$$

Definition 8. *A NGS scheme is* optimally opening sound *if* $Adv^{OS}_{NGS,\mathcal{A}}(\lambda)$ *is negligible for any* $\mathcal{A}$ *and* λ*.*

Opening coherence Opening coherence ensures that the user identified via an opening is the only one able to trace the associated nickname. In this experiment, the issuer is honest while the opener can be malicious. The goal of the adversary is to produce a nickname nk^* that an honest user can trace, and an opening proof that points to another user. It is given the same oracles as in Section 3.3.

We define the advantage in the opening-coherence experiment described in Figure 3 for any polynomial time adversary $\mathcal{A}$ as:

Definition 9. *A NGS scheme is* opening coherent *if* $Adv^{OC}_{NGS,\mathcal{A}}(\lambda)$ *is negligible, for any* $\mathcal{A} = \mathcal{A}^{SndToU, USK, Sig}$ *and any* λ*.*

Anonymity Anonymity ensures that no adversary can identify a specific signer from a target nickname nk^* and signature σ^* pair. We consider IND-CPA security and thus don't provide the *Open* and $Trace$ oracles. Moreover, the *Sig* oracle implicitly offering the tracing capability, we define and provide instead the *SigR* oracle that sign a message for a certain user on a nickname honestly generated from the random coin provided by $\mathcal{A}$. In this experiment, the issuer is corrupted, i.e, the adversary $\mathcal{A}$ is given the issuer key, but not the opener's one. He can use the *SndToU* oracle to add an honest user to the group and write to the registry with the *WReg* oracle. He has also access to the Ch_b oracle, providing a way, given two honest users and a message, to receive a signature from one of the two; note that Ch_b simulates an honest sender that calls the $Nick$ function. He can also call the *USK* oracle to get the secret keys of all users, except the ones that are challenged in the Ch_b oracles (for which the oracles check that their secret keys have not been exposed). Finally, he can also call the *SigR* oracle for user i, random coin r and message m of his choice.

We define the advantage in the anonymity experiment described in Figure 3 for any polynomial time adversary

$$\mathcal{A} = \mathcal{A}^{SndToU, USK, WReg, SigR, Ch_b}$$

as $Adv^{Anon}_{NGS,\mathcal{A}}(\lambda) = |\Pr[Exp^{Anon-0}_{NGS,\mathcal{A}}(\lambda) = 1] - \Pr[Exp^{Anon-1}_{NGS,\mathcal{A}}(\lambda) = 1]|$.

[4] As for traceability, a similar security property could be defined for *Trace*, and proven along the same lines as here.

Definition 10. *A NGS scheme is* anonymous *if $Adv^{Anon}_{NGS,\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and any λ.*

4 Implementation

As stated in the introduction, the NGS implementation we propose is based on GS from PS signatures [27], in particular [24], compatible with our application domain (see Appendix D for an example). During the NGS joining phase, the "issuer" computes a PS signature on the user i's committed key, and saves it (resp., the user encrypted trapdoor for the opener) in **mpk** (resp., **reg**). Anyone can then generate a new nickname nk for this new member by randomizing **mpk**$[i]$, and the GVf verification function reduces to checking the validity of PS signatures. Using his trapdoor, member i can trace such a new nickname and prove possession of it by performing a signature proof of knowledge on a message of his choice. The opener decrypts all trapdoors to check which user's equivalence class identifies a member from a given nickname. The notations for signatures and proofs of knowledge are provided in Appendices A and B.

4.1 Types and variables

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups, with $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, a pairing map. Master public keys are arbitrary representations of users' equivalence classes; they therefore are of the same nature as nicknames. Here they belong to $\mathbb{G}_1^3$, and thus $l = 3$ in $\mathcal{R}$. We assume that g, resp. $\hat{g}$, is a generator of $\mathbb{G}_1$, resp. $\mathbb{G}_2$.

A join request $reqU$ consists of a tuple $(f, w, \tau', \pi_J, \sigma_{DS})$ with $(f, w) \in \mathbb{G}_1^2$, $\tau' \in \mathbb{G}_2^2$ (the El-Gamal encryption of a trapdoor τ), π_J, the proof of knowledge for PK_J computed during the $Join$ operation, and σ_{DS}, a DS signature, for some Digital Signature scheme DS.

A registration information reg consists of a tuple $(f, \tau', \rho, \sigma_{DS})$, with $f \in \mathbb{G}_1$, $\tau' \in \mathbb{G}_2^2$, $\rho \in \mathbb{G}_T$ and σ_{DS}, a DS signature.

Finally, let $H : \mathbb{G}_1 \to \mathbb{G}_1$ be a hash function.

4.2 Functions

Our scheme implements the functions defined in Section 3 as follows.

- $IKg : 1^\lambda \to (isk, ipk)$. Select $(x, y) \leftarrow_\$ \mathbb{Z}_p^2$, and compute $\hat{X} = \hat{g}^x$ and $\hat{Y} = \hat{g}^y$. Return $((x, y), (\hat{X}, \hat{Y}))$.
- $OKg : 1^\lambda \to (osk, opk)$. Select $z \leftarrow_\$ \mathbb{Z}_p$. Return $(z, \hat{g}^z)$.
- $UKg : 1^\lambda \to (usk, upk)$. Return $DS.KeyGen(1^\lambda)$.
- $Join : (usk, opk) \to (msk, \tau, reqU)$.
  1. Select $(\alpha, s) \leftarrow_\$ \mathbb{Z}_p^2$, and compute $f = g^\alpha, u = H(f)$ and $w = u^\alpha$. Let $\hat{Z} = opk$. Then, compute $\tau = \hat{g}^\alpha$ and its El-Gamal encryption $\tau' = (\hat{S}, \hat{f}')$, with some s such that $\hat{S} = \hat{g}^s$ and $\hat{f}' = \tau \cdot \hat{Z}^s$.

   2. Generate a proof π_J for all the above definitions, with $\pi_J = PK_J\{(\alpha, s) : f = g^\alpha \wedge w = u^\alpha \wedge \hat{S} = \hat{g}^s \wedge \hat{f}' = \hat{g}^\alpha \cdot \hat{Z}^s\}$.
   3. Finally, let $\sigma_{DS} = DS.Sig(usk, \rho)$, where $\rho = e(f, \hat{g})$, $msk = \alpha$ and $reqU = (f, w, \tau', \pi_J, \sigma_{DS})$. Then, return $(msk, \tau, reqU)$.
- $Iss : (i, isk, reqU, opk) \to ()$ or $\perp$. Let $(f, w, \tau', \pi_J, \sigma_{DS}) = reqU$ and $(x, y) = isk$. Compute $u = H(f)$ and $\rho = e(f, \hat{g})$ and check the following (return $\perp$ if this fails): (1) f did not appear previously; (2) π_J is valid for $\hat{Z} = opk$; (3) σ_{DS} is valid on ρ under $\mathbf{upk}[i]$. Then compute $v = u^x \cdot w^y$ and set $\mathbf{reg}[i] = (f, \tau', \rho, \sigma_{DS})$ and $\mathbf{mpk}[i] = (u, v, w)$. Return $()$.
- $Nick : (mpk, r) \to nk$. Let $(u, v, w) = mpk$. Then, let $nk = (u^r, v^r, w^r)$. Return nk.
- $Trace : (ipk, \tau, nk) \to b$. Let $(u, v, w) = nk$ and $b = (e(u, \tau) = e(w, \hat{g})) \wedge GVf(ipk, nk)$. Return b.
- $Sign : (nk, msk, m) \to \sigma$. Let $(u, v, w) = nk$ and $\alpha = msk$. Compute and return $\sigma = SPK_S\{(\alpha) : w = u^\alpha\}(m)$.
- $GVf : (ipk, nk) \to b$. Let $(u, v, w) = nk$ and $(\hat{X}, \hat{Y}) = ipk$. Compute and return $b = (e(v, \hat{g}) = e(u, \hat{X}) \cdot e(w, \hat{Y}))$.
- $UVf : (nk, m, \sigma) \to b$. Let $(u, v, w) = nk$ and return $b = true$ if the signature proof of knowledge σ is valid with respect to (u, w) and m, *false* otherwise.
- $Open : (osk, nk) \to (i, \Pi)$ or $\perp$. Let $(u, v, w) = nk$ and $z = osk$.
   1. For each $reg = (f, \tau', \rho, \sigma_{DS}) \in \mathbf{reg}$:
      (a) let $(\hat{S}, \hat{f}') = \tau'$, and get the user trapdoor τ as $\tau = \hat{f}' \cdot \hat{S}^{-z}$;
      (b) check if $(e(u, \tau) = e(w, \hat{g}) \wedge \rho = e(g, \tau))$.
   2. If Step 1 fails for all reg, then output $\perp$, and, otherwise, let i the index of the (unique) reg that succeeds;
   3. Compute $\pi_O = PK_O\{(\tau) : e(w, \hat{g}) = e(u, \tau) \wedge \rho = e(g, \tau)\}$;
   4. With $\Pi = (\rho, \sigma_{DS}, \pi_O)$, return (i, Π).
- $Judge : (nk, i, ipk, \Pi) \to b$. First, let $(\rho, \sigma_{DS}, \pi_O) = \Pi$ and $(u, v, w) = nk$; get $upk = \mathbf{upk}[i]$. Then, check: (1) the validity of π_O; (2) whether $DS.Vf(upk, \rho, \sigma_{DS}) = 1$; (3) the value of $GVf(ipk, nk)$. Output $b = true$, if the three conditions hold, and *false*, otherwise.

That this implementation indeed corresponds to NGS interface (see Section 3.1) can be seen by case analysis. As an example directly related to nicknames, we consider the *Trace* case, i.e., the equivalence between $(nk \in [mpk]_{\mathcal{R}})$, for some $nk = (u', v', w')$ and issuer-generated $mpk = (u, v, w)$, and its implementation $Trace(ipk, \tau, nk)$, i.e. $(e(u', \tau) = e(w', \hat{g})) \wedge GVf(ipk, nk)$, for τ compatible with mpk. The forward direction is easy, since there exists r such that $nk = mpk^r$; checking that $Trace(ipk, \tau, mpk^r)$ is true is straightforward, given e's properties, and how (u, v, w) is constrained in Iss, and τ, in $Join$.

The backward direction assumes the existence of $nk = (u', v', w')$ such that, when passed to *Trace* with some τ, the result is true. Since τ is in $\mathbb{G}_2$, there exists some α, such that $\tau = \hat{g}^\alpha$. Since e is injective, $w' = u'^\alpha$. Then, one can check that v' satisfies the Iss-like constraint $v' = u'^x \cdot w'^y$. Completing the proof that $(nk \in [mpk]_{\mathcal{R}})$ thus just requires finding some r such that $u' = u^r$ (which would also work for v' and w', which only depend on u'). Since both u and u' are in the cyclic group $\mathbb{G}_1$, of order p, there exists i' and i in $\mathbb{Z}_p$ such that $u' = g^{i'}$ and $u = g^i$. It suffices thus to pick r such that $i' \equiv ri \bmod p$, the existence of which is a direct consequence of Bézout's theorem, since p is prime.

4.3 Security analysis

Assuming that the SPKs used above are simulation-sound extractable NIZKs and that H is modeled as a random oracle, then one can prove (see Appendix C), following the definitions in Sections 3.4 and subsequents (see also Appendix B), that this NGS implementation is correct, non-frameable (under the EUF-CMA of the underlying digital signature scheme and the Symmetric Discrete Logarithm assumption), optimally opening sound, opening coherent, traceable (under the Modified GPS assumption [24]) and anonymous (under the Symmetric External Diffie-Hellman assumption, SXDH).

5 Conclusion and Future work

We introduce Nicknames for Group Signatures (NGS), a new signing scheme that combines anonymous transfers from signatures with flexible public keys with auditability from group signatures. We formally define the NGS security model and describe an efficient implementation, which we proved secure in the Random Oracle Model. We developed a full implementation of NGS in Rust that we used in NickHat, an auditable, compliant solution for any ERC-20-based token in Ethereum.

Our NGS implementation focus on efficiency for our Ethereum application. Regarding future work, one might want to enforce stronger security properties, in particular the anonymity and traceability. One could also want to take advantage of the "batching property" that comes with Kim et al's implementation [24], from which we borrow, to improve the performance of the GVf function. Finally, the issue of user revocation needs to be addressed, e.g., in the line of [25].

References

1. M. Abe, S. S. Chow, K. Haralambiev, and M. Ohkubo. Double-trapdoor anonymous tags for traceable signatures. *Int. J. of information security*, 12:19–31, 2013.
2. M. Backes, L. Hanzlik, K. Kluczniak, and J. Schneider. Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys, 2018. Revision of an ASIACRYPT'18 pub.
3. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT'03, Proc. 22*, pages 614–629. Springer, 2003.
4. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *Cryptographers' track, RSA Conf.*, pages 136–153. Springer, 2005.

5. D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *ASIACRYPT'12: Beijing, China, Proc. 18*, pages 626–643. Springer, 2012.
6. P. Bichsel, J. Camenisch, G. Neven, N. Smart, and B. Warinschi. Get shorty via group signatures without encryption. In *Security and Cryptography for Networks - SCN 2010*, volume 6280, pages 381–398. Springer, 2010.
7. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *Proc. of the 11th ACM Conf. on CCS*, pages 168–177, 2004.
8. V. Buterin. Ethereum white paper, 2014.
9. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Eurocrypt'01*, pages 93–118. Springer, 2001.
10. D. Chaum. Blind signatures for untraceable payments. In *Ann. Int. Cryptology Conf.*, 1982.
11. D. Chaum and E. Van Heyst. Group signatures. In *EUROCRYPT'91, Brighton, UK, Proc. 10*, pages 257–265. Springer, 1991.
12. F. Chiacchio, D. D'Urso, L. M. Oliveri, A. Spitaleri, C. Spampinato, and D. Giordano. A non-fungible token solution for the track and trace of pharmaceutical supply chain. *Applied Sciences*, 12(8):4019, 2022.
13. D. Derler and D. Slamanig. Highly-efficient fully-anonymous dynamic group signatures. In *Proc. of the Asia Conf. on CCS*, pages 551–565, 2018.
14. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO' 86*, pages 186–194. Springer, 1987.
15. E. Fujisaki and K. Suzuki. Traceable ring signature. In *Int. Workshop on Public Key Cryptography*, pages 181–200. Springer, 2007.
16. GDPR. General data protection regulation. 2016.
17. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. on computing*, 17(2):281–308, 1988.
18. J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *ASIACRYPT'06*, pages 444–459. Springer, 2006.
19. C. Hanser and D. Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In *ASIACRYPT'14, Kaoshiung, Taiwan, ROC, Proc., Part I 20*, pages 491–511. Springer, 2014.
20. C. Hébant and D. Pointcheval. Traceable constant-size multi-authority credentials. *Information & Computation*, 293, 2023.
21. D. Hopwood, S. Bowe, T. Hornby, N. Wilcox, et al. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 4(220):32, 2016.
22. A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *Asiacrypt'04*, pages 571–589. Springer, 2004.
23. A. Kiayias, Y. Tsiounis, and M. Yung. Group encryption. In *Asiacrypt'07*, pages 181–199. Springer, 2007.
24. H. Kim, Y. Lee, M. Abdalla, and J. H. Park. Practical dynamic group signature with efficient concurrent joins and batch verifications. *J. of information security and applications*, 63, 2021.
25. B. Libert, T. Peters, and M. Yung. Scalable group signatures with revocation. In *EUROCRYPT'12*, pages 609–627. Springer, 2012.
26. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography: 6th Ann. Int. Workshop, SAC'99 Kingston, Ontario, Canada, 1999 Proc. 6*, pages 184–199. Springer, 2000.

27. D. Pointcheval and O. Sanders. Short randomizable signatures. In *Topics in Cryptology - CT-RSA 2016*, pages 111–126. Springer, 2016.
28. G. Quispe, P. Jouvelot, and G. Memmi. Nickpay, an auditable, privacy-preserving, nickname-based payment system. In *to appear*. IEEE, 2025.
29. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT'01, Gold Coast, Australia, Proc. 7*, pages 552–565. Springer, 2001.
30. V. Valaštín, K. Košt'ál, R. Bencel, and I. Kotuliak. Blockchain based car-sharing platform. In *2019 International Symposium ELMAR*, pages 5–8. IEEE, 2019.
31. H. Wang, Z. Yu, Y. Liu, B. Guo, L. Wang, and H. Cui. Crowdchain: A location preserve anonymous payment system based on permissioned blockchain. In *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 227–233. IEEE, 2019.

A Digital signature

Definition 11 (Digital signature). *A* digital signature scheme *(DS) consists of 3 algorithms:*

- *KeyGen* $: 1^\lambda \to (sk, pk)$*, a key generation algorithm that outputs a pair of secret and public keys* (sk, pk) *under security parameter* λ*;*
- *Sig* $: (sk, m) \to \sigma$*, a signing algorithm taking a secret key sk and a message* $m \in \{0,1\}^*$ *and yielding the signature* σ *for* m*;*
- *Vf* $: (pk, m, \sigma) \to \{0,1\}$*, a verification algorithm that takes a signature* σ*, a message* m *and a public key pk and outputs 1, if* (m, σ) *is valid (i.e., the secret key used to build* σ *and the public key pk match under DS) and 0, otherwise.*

The standard security property for a digital signature scheme is "existential unforgeability under chosen message attacks" (EUF-CMA) [17].

B Proof protocols

We recall the notion of Σ protocols and the notation from [6]. Let $\phi : \mathbb{H}_1 \to \mathbb{H}_2$ be an homomorphism with $\mathbb{H}_1$ and $\mathbb{H}_2$ being two groups of order q and let $y \in \mathbb{H}_2$.

We denote by $PK\{(x) : y = \phi(x)\}$ the Σ protocol for a zero-knowledge proof of knowledge of an x such that $y = \phi(x)$. Σ protocols are three-move protocols between a prover P and a verifier V described as follows.

1. $P \rightsquigarrow V$: P chooses $rnd \leftarrow_\$ \mathbb{H}_1$ and sends $Comm = \phi(rnd)$ to V, where $\leftarrow_\$$ denotes the uniform-sampling operation on a finite set, here on $\mathbb{H}_1$.
2. $V \rightsquigarrow P$: Once $Comm$ is received, V chooses a challenge $Cha \leftarrow_\$ \mathbb{H}_1$ and sends it to P.
3. $P \rightsquigarrow V$: P computes $Rsp = rnd - Cha \cdot x$, and sends it to V who checks whether $(\phi(Rsp) \cdot \phi(x)^{Cha} = Comm)$ or not.

We denote by $\pi = SPK\{(x) : y = \phi(x)\}(m)$ with $m \in \{0,1\}^*$ the signature variant of a Σ protocol obtained by applying the Fiat-Shamir heuristic ([14, 5]) to it; this is called the "signature proof of knowledge" on a message m. The Fiat-Shamir heuristic removes the interaction by calling a "random oracle" to be used in security proofs, instantiated by a suitable hash function $H : \{0,1\}^* \to \mathbb{Z}_q$.

1. $P \rightsquigarrow V$: P chooses $rnd \leftarrow_{\$} \mathbb{H}_1$, computes $Cha = H(\phi \parallel y \parallel \phi(rnd) \parallel m)$ and $Rsp = rnd - Cha \cdot x$, and sends (Cha, Rsp) to V.
2. V: V accepts (Cha, Rsp) iff $(Cha = H(\phi \parallel y \parallel \| y^{Cha} \cdot \phi(Rsp) \parallel m))$.

where $\|$ is the string-concatenation function on binary numbers (we assume a proper binary encoding of the ϕ function and of elements of $\mathbb{H}_2$).

The security properties of SPKs, previously defined in [5] and [18], can be informally described as follows.

- "Completeness" states that a signature generated by an honest signer should be verified successfully.
- "Zero-knowledge" (ZK) ensures that a zero-knowledge simulator $\mathcal{S}$ able to simulate a valid proof, a SPK, without knowing the witness x and indistinguishable from a real one, does exist.
- "Simulation soundness" (SS) states that a malicious signer with no witness is unable to generate a proof for a false statement (even receiving simulated proofs).
- "Simulation-sound extractability" (SE) ensures that there exists a knowledge extractor $\mathcal{E}$ able to extract a correct witness from a valid proof generated by a malicious signer.

Definition 12 (Simulation-sound extractable SPK). *A protocol is a* simulation-sound extractable *non-interactive zero-knowledge (NIZK) signature variant of a Σ protocol (SPK) if it satisfies completeness, zero-knowledge, and simulation-sound extractability.*

C Proofs

C.1 Traceability proof

$\mathcal{C}$ is given an instance of the Modified GPS problem, i.e., a pair $(\hat{X}, \hat{Y})$ such that their respective discrete logarithms (x, y) are unknown to him, and the two oracles $\mathcal{O}_0^{MGPS}$ and $\mathcal{O}_1^{MGPS}$. $\mathcal{C}$ acts as an honest issuer and maintains the lists L_0^{MGPS} and L_1^{MGPS} for $\mathcal{O}_0^{MGPS}$ and $\mathcal{O}_1^{MGPS}$, respectively, storing input/output values. Also he can use $\mathcal{E}$, the extractor of PK_S.

$\mathcal{C}$ sets $ipk = (\hat{X}, \hat{Y})$ and samples $osk = (z_0, z_1) \leftarrow_{\$} \mathbb{Z}_p^2$. It then answers the different oracles queries as follows, assuming that $\mathcal{A}$ has access to ipk and osk ($CrptU$, USK and $RReg$ operate as already specified, and L_H is an initially empty mapping for hashed values).

- Hash. Given input n such that $(n, u) \in L_H$ for some u, return u. Otherwise, if $n \in \mathbb{G}_1$, $\mathcal{C}$ calls $\mathcal{O}_0^{MGPS}$ to get some $u \in \mathbb{G}_1$ and adds (n, u) to L_H and u to L_0^{MGPS}. Otherwise, $\mathcal{C}$ samples $u \leftarrow_\$ \mathbb{G}_1$ and add (n, u) into L_H. Finally, u is returned.
- AddU. $\mathcal{C}$ follows the *Addu* protocol to add the user i. However, since it does not know isk, v is obtained by calling $\mathcal{O}_1^{MGPS}$ (L_1^{MGPS} gets thus updated) with (g, u, f, w) as input computed following the protocol. Finally, (i, τ) is added to L_h.
- SndToI: For a queried identity i such that $(i, \mathit{cont}) \in L_c$ (i.e., i comes from *CrptU*; otherwise $\perp$ is returned), $\mathcal{C}$ also receives its join request $r = (f, w, \tau', \pi_J, \sigma_{DS})$. $\mathcal{C}$ checks that f is unique (i.e., not in **reg**), that σ_{DS} is a valid signature for the message $\rho = e(f, \hat{g})$ and public key $\mathbf{upk}[i]$, and that π_J is also valid. $\mathcal{C}$ then computes $u = Hash(f)$, therefore obtained from $\mathcal{O}_0^{MGPS}$. As in the *CrptU* oracle, $\mathcal{C}$ does not know isk and calls $\mathcal{O}_1^{MGPS}$ with (g, u, f, w) as input to get $v = u^x \cdot w^y$; L_1^{MPGS} is, similarly, updated after the call to $\mathcal{O}_1^{MGPS}$. Then, $\mathbf{mpk}[i]$ is set to (u, v, w), and $\mathbf{reg}[i]$, to $(i, \tau', \rho, \sigma_{DS})$. L_c is updated with (i, accept).

At the end, $\mathcal{A}$ outputs its forgery $nk^* = (u^*, v^*, w^*)$, σ^* and m^* with σ^* valid on the message m^*. Now, as $GVf(nk^*)$ and σ^* should be valid, as stated in the experiment, two cases are possible.

- $Open(osk, nk) = \perp$. We make the contradiction explicit. For each $reg = (i, \tau', \rho, \sigma_{DS}) \in \mathbf{reg}$, $\mathcal{C}$ decrypts τ' to get τ. In this case, for each reg, at least one of the two pairing check fails, i.e., $e(u^*, \tau) \neq e(w^*, \hat{g})$ or $\rho \neq e(g, \tau)$. The second inequality would break the simulation soundness of PK_J and is therefore excluded.
  Thus, $w^* \neq {u^*}^{\alpha}$ for each user's α such that $\tau = \hat{g}^\alpha$. Therefore, $\alpha^* = \log_{u^*} w^*$ is new, i.e., not stored in **reg**. But, then, $\mathcal{C}$ could use $\mathcal{E}$ of PK_S on σ^* to extract this α^* and solve the MGPS problem with $((u^*, v^*), \alpha^*)$, a contradiction.
- There exist i^* and Π^* with $(i^*, \Pi^*) = Open(osk, nk)$ and $Judge(nk^*, ipk, i^*, \Pi^*)$ is false. As the *Open* algorithm is run honestly, this case indicates that the τ^* encrypted in $\mathbf{reg}[i^*]$ satisfies both $e(u^*, \tau^*) = e(w^*, \hat{g})$ and $\rho = e(g, \tau^*)$. Thus, since the proof π_O holds with these values and σ_{DS} on ρ has already been verified in the *SndToI* oracle, *Judge* should be true, a contradiction.

C.2 Non-frameability proof

Let $(g, \hat{g}, D, \hat{D})$, with $D = g^d$ and $\hat{D} = \hat{g}^d$, be an instance of the SDL (Symmetric Discrete-Logarithm) problem for some unknown d, and let $\mathcal{S}$ be a zero-knowledge simulator and $\mathcal{E}$, a knowledge extractor for PKs. Let q be the number of queries to the *SndToU* oracle to be made by $\mathcal{A}$. The challenger $\mathcal{C}$ picks a random $k \in \{1, .., q\}$ and hopes that the target user i^* corresponds to the identity of the k^{th} query.

We describe below how the challenger $\mathcal{C}$ responds to the relevant oracles. It uses a caching table L_H, initialized with the tuple (D, δ), where $\delta \leftarrow_\$ \mathbb{Z}_p$.

- Hash: For the queried input n, if $(n, \delta) \in L_H$, $\mathcal{C}$ returns g^δ. Otherwise, $\mathcal{C}$ samples $\delta \leftarrow_\$ \mathbb{Z}_p$, stores (n, δ) in L_H and returns g^δ.
- SndToU: For the queried identity i, if i hasn't been queried before, $\mathcal{C}$ runs UKg to get its (usk, upk); otherwise, $(usk, upk) = (\mathbf{usk}[i], \mathbf{upk}[i])$. Then, if this is not the k^{th} query, $\mathcal{C}$ completes the original *SndToU*. Otherwise, $\mathcal{C}$ sets $i^* = i$, samples $(s_0, s_1) \leftarrow_\$ \mathbb{Z}_p^2$ and gets the value $(D, \delta) \in L_H$. With $(\hat{Z} = opk$, it sets $\mathbf{reqU}[i*] = reqU$ with $reqU = (D, D^\delta, \tau', \pi, \sigma_{DS})$, where $\sigma_{DS} = DS.Sig(\mathbf{usk}[i^*], \rho)$, τ' is the encryption pair $(\hat{g}^s, \hat{D}\hat{Z}^s)$ of $\tau = \hat{D}$, $\rho = e(D, \hat{g})$ and π is a simulated proof of knowledge of the unknown d and of (s_0, s_1), produced by $\mathcal{S}$. Finally, $\mathcal{C}$ returns $reqU$ and adds (i, τ) to L_h.
- USK: for a queried identity i, if $i \notin L_h$ or $i = i^*$, i.e., *SndToU* has been called at least k times, then $\mathcal{C}$ aborts. Otherwise, it sends $(\mathbf{usk}[i], \mathbf{msk}[i])$ to $\mathcal{A}$.
- Sig: for a queried identity i, message m and nickname nk, if $(i, \tau) \notin L_h$ for some τ or $\neg Trace(ipk, \tau, nk)$, then $\mathcal{C}$ returns $\perp$. Otherwise, $\mathcal{C}$ simulates the signature proof of knowledge of $\log_g(\tau)$ on the message m for nk using the simulator $\mathcal{S}$ and returns it.

At the end of the experiment, $\mathcal{A}$ outputs a signing forgery (nk^*, σ^*) on a message m^* with $(u^*, v^*, w^*) = nk^*$ along with the opening result (i, Π^*), with $\Pi^* = (\rho^*, \sigma^*_{DS}, \pi^*_O)$. If $i \neq i^*$, $\mathcal{C}$ aborts. Otherwise, let $(f_k, w_k, \tau'_k, \pi_k, \sigma_k) = \mathbf{reqU}[i^*]$. $\mathcal{C}$ decrypts, with the opening key opk, τ'_k to get τ_k.

When the forgery is accepted by the *Judge* algorithm, π^*_O is valid and σ^*_{DS} is a valid signature of ρ^* for $\mathbf{upk}[i^*]$. We consider the 3 possible cases of forgery, where ρ_k is the pairing signed in σ_k.

- $\rho_k \neq \rho^*$. Since $DS.Vf(\mathbf{upk}[i^*], \rho^*, \sigma^*_{DS})$, this indicates that the signature σ^*_{DS} on the message ρ^*, which the honest user i^* did not sign, is valid. This breaks the unforgeability of DS.
- $e(w^*, \hat{g}) = e(u^*, \tau)$, for some τ different from τ_k. This indicates that the statement required in the *Open* function is false, thus breaking the simulation soundness of PK_O. Indeed, since $\rho_k = \rho^*$, one has $\rho_k = e(g^d, \hat{g}) = e(g, \hat{g}^d) = \rho^* = e(g, \tau^*)$, for some τ^* referenced in π^*_O. Since e is injective in each of its dimensions, one has $\tau^* = \hat{g}^d = \tau_k$ and, from π^*_O, $e(w^*, \hat{g})$ should be equal to $e(u^*, \tau_k)$.
- $e(w^*, \hat{g}) = e(u^*, \tau_k)$. This last case yields $w^* = (u^*)^d$, and $\mathcal{C}$ can use the knowledge extractor $\mathcal{E}$ on σ^* to get the witness d, i.e., $\log_{u^*}(w^*)$, for the SDL challenge, thus solving it.

Otherwise, if $Trace(ipk, \tau_k, nk^*)$ is true, this second case comes down to the last bullet point above, where we can extract the witness d for the SDL challenge.

Overall, if $\mathcal{A}$ succeeds in the NGS non-frameability experiment on our construction with probability $\epsilon = Adv^{Nf}_{NGS,\mathcal{A}}$, then we showed that

$$\epsilon \leq q(Adv^{EUF}_{DS,\mathcal{A}} + Adv^{SS}_{PK_O,\mathcal{A}} + 2Adv^{SDL}_{\mathcal{A}}),$$

with $Adv^{EUF}_{DS,\mathcal{A}}$, the advantage of the adversary in breaking the unforgeability of the DS signature, $Adv^{SS}_{PK_O,\mathcal{A}}$, the advantage of $\mathcal{A}$ in breaking the simulation soundness of PK_O, and $Adv_{\mathcal{A}}^{SDL}$, the advantage of $\mathcal{A}$ in breaking the SDL problem.

C.3 Opening soundness proof

In the join protocol, since the issuer is honest, it therefore prevents two users from having the same α by checking in *SndToI* and *CrptU* that the same first element of $reqU$, f, doesn't appear in previous or current joining sessions. By the soundness of π_J, the encrypted $\tau = \hat{g}^\alpha$ in $\mathbf{reg}[i]$ is also uniquely assigned to user i.

Now, at the end of the experiment, $\mathcal{A}$ outputs $(nk^*, i_0^*, \Pi_0^*, i_1^*, \Pi_1^*)$ with $nk^* = (u, v, w)$ that successfully passes the *GVf* verification, and $\Pi_0^* = (\rho, \sigma_{DS}^*, \pi_O)$ and Π_1^* successfully verified by the *Judge* algorithm for two distinct users, i_0^* and i_1^*, respectively. We now show a contradiction. Let $w = u^\alpha$ for some $\alpha \in \mathbb{Z}_p$. Since (i_0^*, Π_0^*) is accepted by the *Judge* algorithm, $\rho = e(g, \tau_0)$ for some τ_0 linked to i_0^* by σ_{DS}. Then, by the soundness of π_O, it holds that $e(w, \hat{g}) = e(u, \tau_0)$ for the same τ_0. Thus, $\tau_0 = \hat{g}^\alpha$, where α is the exponent that was uniquely assigned to user i_0^* in the join protocol. But, with the same reasoning, α is the exponent that was uniquely assigned to user i_1^* in the join protocol. But we stressed before that the join protocol prevents two users from having the same α exponent; this is therefore a contradiction.

Overall, if $\mathcal{A}$ succeeds in the NGS opening soundness experiment on with above construction with probability $\epsilon = Adv_{NGS,\mathcal{A}}^{OS}$, then we showed that

$$\epsilon \leq Adv_{PK_O,\mathcal{A}}^{SS} + Adv_{PK_J,\mathcal{A}}^{SS},$$

where $Adv_{PK_J,\mathcal{A}}^{SS}$ is the advantage of $\mathcal{A}$ in breaking the simulation soundness of PK_J, and $Adv_{PK_O,\mathcal{A}}^{SS}$, the one of PK_O.

C.4 Opening coherence proof

In the join protocol, since the issuer is honest, it therefore prevents two users from having the same α by checking in *SndToI* and *CrptU* that the same first element of $reqU$, f, doesn't appear in previous or current joining sessions.

Now, at the end of the experiment, $\mathcal{A}$ outputs (nk^*, i^*, Π^*) with $nk^* = (u, v, w)$ that successfully passes the *GVf* verification but also is traced by some honest user i, (i^*, Π^*) successfully verified by the *Judge* algorithm with $\Pi^* = (\rho^*, \sigma_{DS}^*, \pi_O)$ and $i \neq i^*$. We now show a contradiction. Let $w = u^\alpha$ for some $\alpha \in \mathbb{Z}_p$. Since (i^*, Π^*) is accepted by the *Judge* algorithm, $\rho = e(g, \tau^*)$ for some τ^* linked to i^* by σ_{DS}. Then, by the soundness of π_O, it holds that $e(w, \hat{g}) = e(u, \tau^*)$ for the same τ^*. Thus, $\tau^* = \hat{g}^\alpha$, where α is the exponent that was uniquely assigned to user i^* in the join protocol. But user i is honest and the trapdoor τ of user i is the one from the join protocol, thus binding this user to α.

But we stressed before that the join protocol prevents two users from having the same α exponent; this is therefore a contradiction.

Overall, if $\mathcal{A}$ succeeds in the NGS opening coherence experiment on with above construction with probability $\epsilon = Adv^{OC}_{NGS,\mathcal{A}}$, then we showed that

$$\epsilon \leq Adv^{SS}_{PK_O,\mathcal{A}} + Adv^{SS}_{PK_J,\mathcal{A}},$$

.

C.5 Anonymity proof

For this proof, we chain experiments from G_0, the original experiment $Exp^{Anon-0}_{GNS,\mathcal{A}}$, to G_F, a final experiment where the secret key α of some user i_0 is can be replaced by any random key. . We prove indistinguishability between these experiments.

Note that the complete proof requires a subsequent sequence of hybrid experiments from G_F to $Exp^{Anon-1}_{GNS,\mathcal{A}}$, this time, but we omit it as it can be deduced by reversing the first sequence we present. We describe below the experiments with their proofs of indistinguishability.

- G_0 is the original security experiment $Exp^{Anon-0}_{GNS,\mathcal{A}}$ for the target identity i_0.
- G_1 is the same as G_0, except that $\mathcal{C}$ simulates all PKs without witnesses, assuming it can use $\mathcal{S}$, the zero-knowledge simulator of PKs. By the zero-knowledge property of PKs, G_1 is indistinguishable from G_0.
- G_2 is the same as G_1, except that $\mathcal{C}$ aborts when $\mathcal{A}$ produces a proof on a false statement for the PKs. G_2 is indistinguishable from G_1 by the simulation soundness of the PKs.
- G_3 is the same as G_2, except that $\mathcal{C}$ guesses the target identities i_0 and i_1 that will be queried by the first call to Ch_0 oracle. Suppose there are q queries sent to the *SndToU* oracle, $\mathcal{C}$ picks a random pair $(k_0, k_1) \leftarrow_{\$} \{1, .., q\}^2$ and selects the k_0-th and k_1-th queries to the *SndToU* to register, from L_h, the i_0 and i_1 identities. If the guess is wrong, $\mathcal{C}$ aborts. This happens with probability at most $\frac{1}{q^2}$ and thus reduces the advantage by q^2.
- G_4 is the same as G_3, except that, in *SndToU*, $\mathcal{C}$ replaces $\hat{f}'$, in the τ for $\mathbf{reg}[i_0]$, by some random element from $\mathbb{G}_2$. By Lemma 1, G_4 is indistinguishable from G_3.
- G_5 is the same as G_4, except that, in *SndToU*, $\mathcal{C}$ sends a *reqU* request for user i_0 with $\log_u w \neq \log_g f$. By Lemma 2, G_5 is indistinguishable from G_4.
- G_6 is the same as G_5, except that in *SndToU*, $\mathcal{C}$ sends a *reqU* request for user i_1 with $\log_u w \neq \log_g f$. By Lemma 3, G_6 is indistinguishable from G_5.
- G_F is the same as G_6, except that, for (u, w) and (u', w') sent in the *SndToU* oracle for users i_0 and i_1, respectively, we have $\log_u w = \log_{u'} w'$. G_F is indistinguishable from G_6 under Lemma 4.

Lemma 1. *Let H be modeled as a random oracle. Then, G_4 and G_3 are indistinguishable under the SXDH assumption for $\mathbb{G}_2$.*

Given $(\hat{g}, \hat{A}, \hat{B}, T)$, an instance of the SXDH assumption on $\mathbb{G}_2$, where $(\hat{A}, \hat{B}) = (\hat{g}^a, \hat{g}^b)$ for some unknown pair (a, b) and T is given. First, $\mathcal{C}$ runs IKg and sets (isk, ipk) with its return value. Finally, $\mathcal{C}$ sets $opk = \hat{B}$ (osk will not be needed).

- SndToU: if $i \neq i_0$ and $i \neq i_1$, $\mathcal{C}$ follows the default protocol, by calling UKg and $Join$, except that the proof π_J is simulated. Otherwise, $\mathcal{C}$ picks $\alpha \leftarrow_\$ \mathbb{Z}_p$ and simulates π_J with $\mathcal{S}$ for $(f, w, \hat{S}, \hat{f}') = (g^\alpha, H(g^\alpha)^\alpha, \hat{A}, \hat{g}^\alpha \cdot T)$. This tuple along with (π_J, σ_{DS}), where σ_{DS} is a signature on $\rho = e(f, \hat{g})$, are used in the returned $reqU$. At the end, $\mathcal{C}$ sets $\mathbf{msk}[i] = \alpha$ and updates the list of honest users: $L_h = L_h \cup \{(i, \hat{g}^\alpha)\}$.
- SigR: $\mathcal{C}$ executes the oracle algorithm, except that the proof of knowledge of $\mathbf{msk}[i]$ is simulated.

Therefore, $\mathcal{C}$ simulates G_3, if $T = \hat{g}^{ab}$, and G_4, otherwise. Indeed, by definition of π_J, one has $\hat{f}' = \hat{g}^\alpha \cdot opk^s$, where, here, $s = a$ and $opk = \hat{B}$; this yields $\hat{f}' = \hat{g}^\alpha \cdot \hat{g}^{ab}$, which is to be compared with $\hat{g}^\alpha \cdot T$. If $\mathcal{A}$ can distinguish the two experiments, $\mathcal{C}$ could then use it to break SXDH on $\mathbb{G}_2$.

Lemma 2. *Let H be modeled as a random oracle. Then G_5 and G_4 are indistinguishable under the $XDH_{\mathbb{G}_1}$ assumption.*

$\mathcal{C}$ is given (g, A, B, T), with $A = g^a$ and $B = g^b$, an instance of the SXDH assumption on $\mathbb{G}_1$, and the zero-knowledge simulator $\mathcal{S}$ of the SPKs. $\mathcal{C}$ first initializes L_H with $\{(B, \perp, A)\}$.

- Hash: for a queried input n, if some $(n, \delta, \zeta) \in L_H$, $\mathcal{C}$ returns ζ. Else, $\mathcal{C}$ first samples $\delta \leftarrow_\$ \mathbb{Z}_p$ and returns g^δ. He then updates $L_H = L_H \cup \{(n, \delta, \zeta)\}$.
- SndToU: if $i \neq i_0^*$, $\mathcal{C}$ follows the protocol, except that the proof is simulated. Otherwise, $\mathcal{C}$ sets $(f, w, \hat{S}, \hat{f}') = (B, T, \hat{g}^s, \hat{R})$, with $s \leftarrow_\$ \mathbb{Z}_p$ and $\hat{R} \leftarrow_\$ \mathbb{G}_2$. Let π_J be the simulated proof of knowledge of $b = \log_g B$ and s, and σ_{DS}, the DS signature on $\rho = e(B, \hat{g})$. $\mathcal{C}$ stores $\mathbf{msk}[i_0] = \perp$ and updates $L_h = L_h \cup \{(i_0, \perp)\}$. Finally, the request built using these variables as in $Join$ is returned.
- SigR: $\mathcal{C}$ executes the oracle algorithm, except that the proof of knowledge of $\mathbf{msk}[i]$ is simulated.
- Ch$_0$: Let $(x, y) = isk$. For $i = i_0$, $\mathcal{C}$ samples $r \leftarrow \mathbb{Z}_p$ and computes $v = u^x \cdot w^y$, with $(u, w) = (g^r, B^r)$; otherwise, the standard protocol is followed. $\mathcal{C}$ then simulates the proof of knowledge σ for SPK_S of $b = \log_g B$. Finally, $\mathcal{C}$ adds (i_0, m, nk, σ) and (i_1, m, nk, σ) to L_{ch} and returns (nk, σ), with $nk = (u, v, w)$.

Since T has been introduced as w for i_0, $\mathcal{C}$ simulates G_4 if $T = g^{ab}$, and G_5 otherwise. Indeed, if $T = g^{ab}$, we have $f = B, u = H(f) = A$ and $w = u^b = T$, thus implementing a proper join protocol. Note also that Ch_0 has been modified to ensure a proper behavior even if $\mathbf{mpk}[i_0]$ is defined with values given in *SndToU*.

Lemma 3. *Let H be modeled as a random oracle. Then G_5 and G_6 are indistinguishable under the $XDH_{\mathbb{G}_1}$ assumption.*

The proof follows the same strategy as in Lemma 1, since G_5 is indistinguishable from G_4, but for user i_1.

Lemma 4. *Let H be modeled as a random oracle. Then G_6 and G_F are indistinguishable under the XDH$_{\mathbb{G}_1}$ assumption.*

$\mathcal{C}$ is given (g, A, B, T), with $A = g^a$ and $B = g^b$, an instance of the SXDH assumption on $\mathbb{G}_1$, and the zero-knowledge simulator $\mathcal{S}$ of the SPKs.

- Hash: for a queried input n, if n has already been queried, $\mathcal{C}$ returns ζ for the corresponding (n, ζ) in L_H. Otherwise, $\mathcal{C}$ samples $\delta \leftarrow_\$ \mathbb{Z}_p$, updates $L_H = L_H \cup \{(n, g^\delta)\}$, and returns g^δ.
- SndToU: if $i \neq i_0 \wedge i \neq i_1$, $\mathcal{C}$ follows the protocol, except that the proofs are simulated. Otherwise, he first updates $L_h = L_h \cup \{(i, \perp)\}$. Then, if $i = i_0$, $\mathcal{C}$ samples $\alpha_0 \leftarrow_\$ \mathbb{Z}_p$, adds $\{(g^{\alpha_0}, B)\}$ to L_H, sets $(\hat{S}, \hat{f}') = (\hat{g}^s, \hat{R})$, with $s \leftarrow_\$ \mathbb{Z}_p$ and $\hat{R} \leftarrow_\$ \mathbb{G}_2$, and $(f, w, \tau') = (g^{\alpha_0}, T, (\hat{S}, \hat{f}'))$ and returns $(f, w, \tau', \pi_J, \sigma_{DS})$, where π_J is the $\mathcal{S}$-simulated proof of knowledge PK_J for α_0 and s, and σ_{DS} is the DS signature on $\rho = e(g^{\alpha_0}, \hat{g})$. Before returning, $\mathcal{C}$ stores $\mathbf{msk}[i_0] = \alpha_0$.
  Finally, if $i = i_1$, $\mathcal{C}$ samples $(\alpha_1, \delta_1) \leftarrow_\$ \mathbb{Z}_p^2$, adds $\{(g^{\alpha_1}, g^{\delta_1})\}$ to L_H, sets $(\hat{S}, \hat{f}') = (\hat{g}^s, \hat{R})$, with $s \leftarrow_\$ \mathbb{Z}_p$ and $\hat{R} \leftarrow_\$ \mathbb{G}_2$, and $(f, w, \tau') = (g^{\alpha_1}, A^{\delta_1}, (\hat{S}, \hat{f}'))$ and returns $(f, w, \tau', \pi_J, \sigma_{DS})$, where π_J is the $\mathcal{S}$-simulated proof of knowledge PK_J for α_1 and s, and σ_{DS} is the DS signature on $\rho = e(g^{\alpha_1}, \hat{g})$. Finally, $\mathcal{C}$ stores $\mathbf{msk}[i_1] = \alpha_1$.
- SigR: $\mathcal{C}$ executes the oracle algorithm, except that the proof of knowledge of $\mathbf{msk}[i]$ is simulated.
- Ch$_b$: let $(u, v, w) = \mathbf{mpk}[i_b]$ (if undefined, $\mathcal{C}$ returns $\perp$). $\mathcal{C}$ samples $r \leftarrow \mathbb{Z}_p$ and defines $nk = (u^r, v^r, w^r)$. He then $\mathcal{S}$-simulates the proof of knowledge σ for SPK_S. He then updates $L_{ch} = L_{ch} \cup \{(i_0, m, nk, \sigma), (i_1, m, nk, \sigma)\}$ and returns (nk, σ).

For user i_0, however, in the case where $T = g^{ab}$, the secret key α_0' for his mpk (different from α_0, as per experiment G_6) is the unknown $a = \log_g A$, since $u = H(f) = B$ and $w = B^{\alpha_0'}$, which must be equal to T. But, then, a must also be the secret key α_1 of honest user i_1, since, in that case, via *SndToU*, for user i_1, we have $u = H(f) = g^{\delta_1}$ and $w = (g^{\delta_1})^{\alpha_1} = (g^{\delta_1})^a = A^{\delta_1}$. In this case, $\mathcal{C}$ simulates G_F; otherwise, the target users have different secret keys, and $\mathcal{C}$ simulates G_6.

D NickHat

We introduce NickHat, an Ethereum-based permissioned token-exchange system that supports auditing and provides anonymity (its Solidity-based implementation is not described here, for lack of space). As in [21], NickHat supports two address types: the transparent ones, which are Ethereum standard addresses, and the private ones, related to nicknames. Typically, one can transfer tokens from a transparent address to a private one (to enter NickHat), from a private address to another private one (within NickHat), and finally, from a private address to a transparent one (thus transferring tokens out of NickHat). Although NickHat could work with any type of Ethereum currency, we focus, in this prototype, on tokens that respect the ERC-20 standard.

Roles in this system are close to NGS's ones. We keep the issuer, verifier, and user denominations, the NGS' opener and judge are renamed "supervisor" and "(external) auditor", respectively. We add a "relayer" role that takes specific transactions (following ERC-2771 standard) as inputs and sends them to the blockchain and an "operator" that manages the system. We describe below the main operations of NickHat.

Setup First, an issuer and a supervisor are chosen by the operator; they run the NGS IKg and Okg functions, respectively, to obtain their keys. In the blockchain environment, the operator deploys four smart contracts:

1. the verifier contract, which embeds the NGS UVf and GVf functions for the chosen issuer;
2. the key-registry contract, which adds master public keys and displays them;
3. the NickHat contract, which provides the *deposit*, *transfer* and *withdraw* functions (see below), keeping a *balances* array up-to-date;
4. the forwarder contract, with the *execute* function that redirects requests to their specified destination after verifying the NGS signature.

User registration When a user wants to join the issuer-managed group and use NickHat, he runs the NGS group-joining algorithm with the issuer and obtains, if successful, his master public key. The issuer gets the user's encrypted trapdoor. Finally, the operator saves the new master public key by sending a transaction to the key-registry contract, which makes it publicly readable.

Deposit Assume user i already holds v units of a deployed ERC-20 token. User i wants to transfer these tokens to group member j (including himself, as a group member) without disclosing the latter's identity. To do this, user i first sends a signed transaction to the ERC-20 token that allows the NickHat contract to dispose of his v tokens. Next, user i gets j's master public key mpk from the key-registry contract and computes a nickname $nk = Nick(mpk, r)$ for j and some random coin r via the NGS algorithm. He then signs a transaction to deposit v tokens to nk and sends it to the *deposit* function of the NickHat contract. This contract spends the ERC-20 token allowance and transfers i's tokens to itself, acting as an escrow. The NickHat *balances* array is then updated by setting v tokens for the specified nickname nk and token address. Finally, the NickHat contract emits an Announcement event with the nickname nk.

Detection All group members listen to the Announcement events from the blockchain and call the *Trace* function to check if some of the new nicknames mentioned there belong to them. If so, they know that some tokens have been kept in escrow for them, which they can use as they wish.

Transfer Assume that group member i has v escrowed tokens linked to one of his nicknames, nk, and wants to sends these tokens to group member j. He first gets j's master public key mpk from the key-registry contract and calls the NGS *Nick* function to obtain a new nickname $nk' = Nick(mpk, r')$ for some random coin r'. Then, he prepares a request r asking to send v tokens from nk

to nk' and signs r with the NGS *Sig* function on the hash value of r. A relayer is then handed r and wraps it to a meta-transaction (ERC-2771) that he signs and sends to the *execute* function of the forwarder contract. The forwarder contract first verifies the NGS signature by calling the *UVf* function of the verifier contract and redirects the call to the *transfer* function of the NickHat contract. This contract then updates *balances* (so that $balances[nk'] = v$ and $balances[nk] = 0$) and emits an Announcement event regarding nk'.

Withdraw User i listens to Announcement events and learns that some nk mentioned is his. To withdraw his escrowed tokens to an Ethereum address a, he must proceed along the following way. A request r is first prepared, stating that the owner of nk sends v tokens to a, and is then handed to a relayer. The relayer wraps the request into a transaction and sends it to the *execute* function of the forwarder contract. This contract (1) verifies the signature by calling *UVf*, of the verifier contract, (2) redirects the call to the NickHat *Withdraw* function, which subtracts v tokens from $balances[nk]$ (if possible), and (3) asks the ERC-20 token contract to transfer v tokens from the NickHat address to a.

Audit The supervisor can inspect any nickname announced in blockchain Announcement events. He can retrieve the identity of the user behind a nickname by calling the first part of the NGS *Open* algorithm. In the case of an audit request on a particular nickname nk, the supervisor can prove to the external auditor that a certain group member is behind nk. The external auditor can run the *Judge* function to be convinced.

That this implementation indeed corresponds to the previous interface specification can be seen by case analysis. As an example directly related to nicknames, we consider the *Trace* case, i.e., the equivalence between $(nk \in [mpk]_{\mathcal{R}})$, for some $nk = (u', v', w')$ and issuer-generated $mpk = (u, v, w)$, and its implementation $Trace(ipk, \tau, nk)$, i.e. $(e(u', \tau) = e(w', \hat{g})) \wedge GVf(ipk, nk)$, for τ compatible with mpk. The forward direction is easy, since there exists r such that $nk = mpk^r$; checking that $Trace(ipk, \tau, mpk^r)$ is true is straightforward, given e's properties, and how (u, v, w) is constrained in Iss, and τ, in $Join$.

The backward direction assumes the existence of $nk = (u', v', w')$ such that, when passed to *Trace* with some τ, the result is true. Since τ is in $\mathbb{G}_2$, there exists some α, such that $\tau = \hat{g}^\alpha$. Since e is injective, $w' = u'^\alpha$. Then, one can check that v' satisfies the Iss-like constraint $v' = u'^x \cdot w'^y$. Completing the proof that $(nk \in [mpk]_{\mathcal{R}})$ thus just requires finding some r such that $u' = u^r$ (which would also work for v' and w', which only depend on u'). Since both u and u' are in the cyclic group $\mathbb{G}_1$, of order p, there exists i' and i in $\mathbb{Z}_p$ such that $u' = g^{i'}$ and $u = g^i$. It suffices thus to pick r such that $i' \equiv ri \bmod p$, the existence of which is a direct consequence of Bézout's theorem, since p is prime.