Nicknames for Group Signatures

Guillaume Quispe[a], Pierre Jouvelot[b,a], Gérard Memmi[a]
[a]LTCI, Télécom Paris, IP Paris, Palaiseau, France
[b]Mines Paris, PSL University, Paris, France

09-13-2024 version

Abstract

With group signatures, a signer can prove membership of a group controlled by a manager, while only an opener is able to precisely 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.
Building upon these two approaches, we propose a novel scheme, *nicknames for group signatures*, where group members now expose their flexible public keys, named master public keys.

Keywords: Compliance; privacy; auditability; group signature; nicknames; stealth.

1 Introduction

Group Signature (GS) is a message-signing scheme introduced by D. Chaum and E. van Heyst in [9]. Such a scheme provides a relative anonymity to the signers (called "group members") as the signature reveals to the public the sole name of his group, and only group members can sign messages. But, it also offers an opening feature, since a group manager [7] can open signatures, i.e., identify a signer based on its signature and thus break anonymity (usually for himself or other stakeholders such an auditing authority). The Dynamic Group Signature scheme (DGS) extends GS so that the group manager has the ability to accept users to join the group at any moment.

Meanwhile, the Signature with Flexible Public Key (SFPK) [2] scheme partitions the key space into equivalence classes induced by a relation $\mathcal{R}$. In other words, one can transform a signer's public key into a different representative of the same equivalence class, i.e., the old and new public keys are related by $\mathcal{R}$, without the help of its secret key. Additionally, its class-hiding property states that, without a so-called "trapdoor", it should not be possible to determine whether two keys belong to the same class. Implicitly, owning such a trapdoor would imply user traceability as the signer could determine whether a public key belongs to his equivalence class and, thus, recover its signing key. Hence, the primitive of key transformation within equivalence classes offers a way to communicate anonymously, as demonstrated in multi-user settings [2].

In this paper, we introduce a new scheme, called Nickname for Group Signature (NGS), which merges concepts from GS and SFPK in order to provide both anonymity and auditability within a communication system. From GS, it inherits the group-management policy where only users accepted by the group manager can be publicly successfully verified. It also provides a communication feature inspired by SFPK, where members are publicly identified by an arbitrary representative of their equivalence class that we call "master public key", noted *mpk*. Anyone can then derive a key, that we call "nickname", from a group member's master public key. Inherited from GS or SFPK, NGS provides anonymity, in that it hides the link between members and their nicknames. It also supports the user traceability property of SFPK, which allows a group member to retrieve his nicknames given a trapdoor, leading to a weakened version of anonymity, called "selfless anonymity" [8]. Finally, NGS provides a GS-opening-like feature, where an opener is able to provably identify a member from any of its nicknames, thus offering auditablity assurance for authorized participants. NGS can thus be viewed as an extension of group signatures with communication capability between users, *à la* SFPK.

Implementation The design of NGS is based on group signatures, which themselves come from randomizable signatures [7][16][15]. This line of work led to DGS, within which two signing steps are performed in the "group joining" phase: first, the group manager computes a randomizable signature on a committed key given by user, and then the user can randomize this signature and prove knowledge of the committed key [17]. A central idea with NGS is to decouple these two signature-building steps and use the result of the first one, randomizable, as the flexible public key of the second. This decoupling allows, in some loose sense, the construction of SFPK from GS.

In more details, during the NGS joining phase, an "issuer" grants the user an equivalence class of $\mathcal{R}$ by signing user i's committed key as usual. But he also publishes this signature as i's master public key mpk_i, stored in a public array $\mathbf{pk}$, and receives and keeps the user's trapdoor. This mpk_i corresponds to an arbitrary representative of the equivalence class $[mpk_i]_{\mathcal{R}}$. The usual group signing/verifying protocol is thus split into two parts. The first one corresponds to the randomization of issuer-provided signatures, to ensure anonymity. In NGS terminology, this means that a nickname nk is derived from the user's master public key mpk_i. A key difference with previous schemes is that the randomization of mpk_i can be done by user i himself or anyone else, as in [2][12], hence the term of "nickname". The associated verification step can then subsequently check if a given nickname nk belongs to the group formed by the set of nicknames of any user. This second part consists of proving that the user controls, i.e., owns, a nickname nk with its own verification function.

We also use the selfless anonymity property [8] of DGS to allow a user to retrieve all of his nicknames thanks to his trapdoor. The opening of a nickname can then be viewed as an iteration over all users' trapdoors.

Contributions In this paper, we present the following contributions:

- the formal definition of the new Nicknames for Group Signatures scheme (NGS);
- a practical implementation of NGS, based on cyclic groups, pairings and zero-knowledge proofs;
- a thorough analysis of NGS security properties (correctness, non-frameability, traceability, opening soundness, and selfless anonymity).

Organization After the introduction in Section 1, Section 2 presents the building blocks used for constructing NGS. In Section 3, we introduce NGS along with a possible implementation based on [15]. We then describe in Section 4 its security model with the associated security proofs. Section 6 covers the related work, while Section 7 offers some perspectives for possible future work. We conclude in Section 7.

2 Background

This section presents the notations and building blocks used in the definition and the implementation of NGS (see section 3.3).

2.1 Bilinear groups

For cryptographic purposes and following [7], our NGS scheme is using three cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p equipped with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- for all $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$ and $(a, b) \in \mathbb{Z}^2$, $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$;
- for all $g \neq 1_{\mathbb{G}_1}$ and $\tilde{g} \neq 1_{\mathbb{G}_2}$, $e(g, \tilde{g}) \neq 1_{\mathbb{G}_T}$;
- the map e is injective for each dimension;
- the map e is efficiently computable, i.e., of non-exponential time complexity.

Note that we use multiplicative notations for cyclic-group operations. We use below type-3 pairings $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, i.e., where $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ in either direction.

2.2 Signature schemes

There exist many signature schemes. We introduce below the ones that are pertinent for the definition of NGS: digital signature [1], group signature [5], and signature with flexible public key [2].

2.2.1 Digital signature

Definition 2.1 (Digital signature) *A* digital signature scheme *(DS) consists of 3 algorithms:*

- *KeyGen* $: 1^\lambda \rightarrow (sk, pk)$*, a key generation algorithm that outputs a pair of secret and public keys* (sk, pk) *under security parameter* λ*;*

- *Sig* $: (sk, m) \rightarrow \sigma$*, a signing algorithm that takes a secret key sk and a message* $m \in \{0, 1\}^*$ *and outputs the signature* σ *for m;*

- *Vf* $: (pk, m, \sigma) \rightarrow \{0, 1\}$*, 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.*

Note that, if only the owner of the pair (sk, pk) can sign a message m that can be properly verified, the verification can be done by anyone having access to the public key. Signatures are designed so that (m, σ) cannot be forged. The standard security property for a digital signature scheme is "existential unforgeability under chosen message attacks" (EUF-CMA) [13]. Informally, it states that, given access to a signing oracle, it is hard (in terms of λ) to output a valid pair (m, σ) for a message m never before submitted to the signing oracle.

2.2.2 Group signature

In group signatures, only the members of a group (also called *signers*), managed by an "issuer" and an "opener", can sign messages on behalf of the group, providing thus anonymity. Moreover, a tracing authority, the *opener*, can revoke the anonymity of the signer. We use the definition from [5] to provide, first, the syntax of group signatures and, then, its security model.

Definition 2.2 (Group signature) *A* group signature *(GS) scheme is a tuple of probabilistic polynomial-time (PPT) algorithms (Setup, IKg, OKg, UKg, GJoin, GSig, GVf, GOpen, GJudge) defined as follows.*

- *Setup* $: 1^\lambda \rightarrow pp$*. Under a security parameter* λ*, the setup algorithm outputs the public parameter pp.*

- *IKg* $: pp \rightarrow (isk, ipk)$*. Given a public parameter pp, the issuer can invoke this key generation algorithm to output the (calling) issuer's secret/public key pair* (isk, ipk)*.*

- $OKg : pp \to (osk, opk)$. *Given a public parameter pp, this opener key-generation algorithm outputs the (calling) opener's secret/public key pair (osk, opk). Note that gpk is used as a shortcut for (ipk, opk), while gsk does likewise for the secret keys.*

- $UKg : (1^\lambda, i) \to (upk_i, usk_i)$. *User i invokes the user key-generation algorithm to produce its public key upk_i and secret key usk_i. We assume that upk_i is authenticated by a Certification Authority (CA).*

- $GJoin : (Join(i, usk_i, gpk) \leftrightarrow Iss(i, upk_i, isk)) \to (gsk_i, reg_i)$. *This interface allows the user i to join the group by an interactive protocol between the user and the issuer. The $Join$ algorithm run by user i takes its secret key usk_i and the group public key gpk. Meanwhile, the Iss algorithm run by the issuer takes its secret key isk and the user's public key upk_i. If successful, i.e., accepted by the issuer, the user receives its group signing key gsk_i, and the issuer adds the user's information reg_i on the registration list, reg. Note that the opener has a read access to reg.*

- $GSig : (gsk_i, m) \to \sigma$. *This function allows user i with group signing key gsk_i to output a signature σ on the message m on behalf of the group.*

- $GVf : (gpk, m, \sigma) \to \{0, 1\}$. *This verification algorithm is publicly available. It takes the group public key gpk, a message m, and a group signature σ of the previous message and outputs 1, if (σ, m) is valid with respect to gpk, and 0, otherwise.*

- $GOpen : (osk, m, \sigma, reg) \to (i, \Pi)$ *or* $\perp$. *This is the algorithm called exclusively by the opener with his opener secret key osk to identify the signer of the signature σ on message m, given the register reg. It outputs the index of the user i with a proof Π stating that user i did produce the signature or $\perp$, if no user has been found.*

- $GJudge : (m, \sigma, gpk, i, upk_i, \Pi) \to \{0, 1\}$. *This publicly available judging algorithm can be used to verify an opener's proof Π. It takes a message m, a signature σ on m, the group public key gpk, an identity i with its public key upk_i, and the opener's proof Π to be verified. It outputs 1, if Π is valid, 0, otherwise.*

As for DS, GS, and also its dynamic variant, support a security model that we very briefly and informally describe, following [5]. We do not provide here a formal description, because the security model of NGS, which we describe below, directly inherits from it, and it would be repetitive to see it described twice.

"Correctness" ensures that a signature from an honest member has to be verified successfully with probability 1. "Anonymity" ensures that no adversary can identify the signer of a target group signature. "Non-framability" ensures that an honest user cannot be falsely accused of having signed a message. "Traceability" ensures that no user can produce a valid group signature that is not traceable by the opener. "Opening soundness" ensures that no adversary can produce a signature that can be opened to two distinct users.

Note that many other security properties such as unforgeability were proposed before, but Bellare et al. [5] showed that their properties, sketched above, encompass these.

2.2.3 Signature flexible public key

In this scheme, the public key of a user belongs to an equivalence class induced by a relation $\mathcal{R}$. Anyone can change it while maintaining it in its equivalence class. Only through the use of a "trapdoor" can one check that a given public key is member of an equivalence class.

Definition 2.3 (Signature with flexible public key) *A* signature scheme with flexible public key *(SFPK) is a tuple of PPT algorithms (KeyGen,TKGen, Sign, ChkRep, ChgPK, Recover, Verify) such that:*

- $KeyGen : (\lambda, \omega) \to (sk, pk)$. *This key-generation algorithm takes as input the security parameter λ and random coins $\omega \in$ Coin and outputs a pair (sk, pk) of secret and public keys.*

- $TKGen : (\lambda, \omega) \to ((sk, pk), \tau)$. *This trapdoor generation algorithm takes as input the security parameter λ and random coins $\omega \in$ Coin and outputs a pair (sk, pk) of secret and public keys with its corresponding trapdoor τ.*

- $Sig : (sk, m) \to \sigma$. *This signing algorithm takes as input a secret key sk and the message to sign m and outputs a signature σ valid for m.*

- $ChkRep : (\tau, pk) \to \{0, 1\}$. *This algorithm takes as input a trapdoor τ for some equivalence class $[pk']_{\mathcal{R}}$ and a public key pk and outputs 1, if $pk \in [pk']_{\mathcal{R}}$, and 0, otherwise.*

- $ChgPK : (pk, r) \to pk'$. *This algorithm takes as input a representative pk of an equivalence class $[pk]_{\mathcal{R}}$ and a random coin r and returns, using r, a different representative pk', where $pk' \in [pk]_{\mathcal{R}}$.*

- $Recover : (sk, \tau, pk) \to sk'$. *This algorithm takes as input a secret key sk, a trapdoor τ, and a representative pk and returns an updated secret sk'.*

- $Verify : (pk, m, \sigma) \to \{0, 1\}$. *This algorithm takes as input a public verification key pk, a message m and a signature σ and outputs 1, if σ is valid for m, and 0, otherwise.*

2.3 Proof protocols

Signature schemes are often based on the need to prove knowledge of discrete logarithms, i.e., powers of members of cyclic groups. To this end, we recall the notion of Σ protocols and the notation from [7]. 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 $y \in \mathbb{H}_2$. For simplicity, we assume below that $\mathbb{H}_1 = \mathbb{Z}_q$.

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 $\texttt{rnd} \leftarrow_{\$} \mathbb{H}_1$ and sends $\texttt{Comm} = \phi(\texttt{rnd})$ to V, where $\leftarrow_{\$}$ denotes the uniform-sampling operation on a finite set, here on $\mathbb{H}_1$.

2. $V \rightsquigarrow P$: Once $\texttt{Comm}$ is received, V chooses a challenge $\texttt{Cha} \leftarrow_{\$} \mathbb{H}_1$ and sends it to P.

3. $P \rightsquigarrow V$: P computes $\texttt{Rsp} = \texttt{rnd} - \texttt{Cha} \cdot x$, and sends it to V who checks whether $(\phi(\texttt{Rsp}) \cdot \phi(x)^{\texttt{Cha}} = \texttt{Comm})$ or not.

Note that we use the same notation (=) for both variable definition and boolean equality; there should be no possible confusion, given the context.

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 ([11, 6]) 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 $\texttt{rnd} \leftarrow_{\$} \mathbb{H}_1$, computes $\texttt{Cha} = H(\phi \parallel y \parallel \phi(\texttt{rnd}) \parallel m)$ and $\texttt{Rsp} = \texttt{rnd} - \texttt{Cha} \cdot x$, and sends $(\texttt{Cha}, \texttt{Rsp})$ to V.

2. V: V accepts $(\texttt{Cha}, \texttt{Rsp})$ iff $(\texttt{Cha} = H(\phi \parallel y \parallel \parallel y^{\texttt{Cha}} \cdot \phi(\texttt{Rsp}) \parallel m))$.

where $\parallel$ 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 [6] and [14] 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 2.4 (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.*

2.4 Complexity assumptions

Existence and impossibility properties used in signature schemes are based on the (assumed) probabilistic hardness of some decision procedures. We present those here.

2.4.1 Decisional Diffie-Hellman

The Decisional Diffie-Hellman (DDH) assumption states that, for any group element g, the probability distributions of (g, g^a, g^b, g^c) and (g, g^a, g^b, g^{ab}), where $(a, b, c) \leftarrow \mathbb{Z}^3$, are computationally indistinguishable. Informally, this implies that, even given the two elements g^a and g^b, the element g^{ab} "seems" as random as any g^c.

The Asymmetric External Decisional Diffie-Hellman (SXDH) assumption extends DDH when dealing with bilinear groups. SXDH assumes DDH for both groups $\mathbb{G}_1$ and $\mathbb{G}_2$, as per Section 2.1.

2.4.2 Symmetric Discrete Logarithm

Considering a type-3 pairing $\Gamma = (\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_{\mathbb{T}}, e)$, the Symmetric Discrete Logarithm (SDL) assumption over $\mathbb{G}$ is that, for any PPT adversary $\mathcal{A}$, for all $g \in \mathbb{G}^*$, $\tilde{g} \in \tilde{\mathbb{G}}^*$ and $x \in \mathbb{Z}_{|}^*$, the probability that $\mathcal{A}$, given $(\Gamma, g, \tilde{g}, g^x, \tilde{g}^x)$ as an input, is able to to return x is negligible.

2.4.3 Pointcheval-Sanders Assumptions

PS assumption [16]. Given a tuple (g^x, g^y) and an oracle $\mathcal{O}^{PS} : m \rightarrow (u, u^{x+ym})$, where $m \in \mathbb{Z}_p$ and $u \in \mathbb{G}_1$ is uniformly distributed, the Pointcheval-Sanders (PS) assumption states that it is impossible to find efficiently, in PPT, a tuple $((u', v'), m) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that $u' \neq 1_{\mathbb{G}_1}$ and $v' = u'^{x+ym}$, while the content of m has not been used, i.e., queried, in any call to $\mathcal{O}^{PS}$.

Generalized PS assumption. Assume given a tuple (g^x, g^y) and two oracles $\mathcal{O}_0^{GPS} : () \rightarrow u$, where $u \in \mathbb{G}_1$ is uniformly distributed, and $\mathcal{O}_1^{GPS} : (u, m) \rightarrow v$, where $u \in \mathbb{G}_1, m \in \mathbb{Z}_p$ and $v = u^{x+ym} \in \mathbb{G}_1$ is output only if u was provided by calling $\mathcal{O}_0^{GPS}$ and was not queried before.

The Generalized PS (GPS) assumption states it is impossible to efficiently find a tuple $((u', v'), m) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that $u' \neq 1_{\mathbb{G}_1}$ and $v' = u'^{x+ym}$, while m has not been queried.

Modified GPS assumption. Assume given a tuple (g^x, g^y) and two oracles $\mathcal{O}_0^{MGPS} : () \rightarrow u$, where $u \in \mathbb{G}_1$ is uniformly distributed, and $\mathcal{O}_1^{MGPS} : (g, u, f, w) \rightarrow v$, where $(g, u, f, w) \in \mathbb{G}_1^4$ and $v = u^x w^y \in \mathbb{G}_1$ is output only if 1) u was not queried to this oracle before, 2) u appeared previously as output of $\mathcal{O}_0^{MGPS}$ and 3) $\log_g f = \log_u w$.

The Modified GPS assumption states that it is impossible to efficiently find a tuple $((u, v), m) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that $u \neq 1_{\mathbb{G}_1}$ and $v = u^{x+ym}$ for some new

m, i.e., such that, for any pair (u', w') part of a previous query to $\mathcal{O}_1^{MGPS}$, $u'^m = w'$ does not hold. Getting u via $\mathcal{O}_0^{MGPS}$, this amounts to finding a value w, part of a query to $\mathcal{O}_1^{MGPS}$, such that the equality $w = u^m$ does not hold, with $m = \log_g f$ for some f.

Note that [15] proved that the last two assumptions hold in the generic group model [18].

3 Nicknames for Group Signatures

We introduce Nicknames for Group Signature (NGS), a new digital signature scheme that adds the concept of "nicknames" on top of the notion of group signatures. Here, we consider, if l denotes the dimension of the space of public keys pk, represented as tuples here, the following equivalence relation $\mathcal{R}$ that partitions $\mathbb{G}^{*l}$, the users' public-key space, into equivalence classes: $\mathcal{R} = \{(pk_1, pk_2) \in (\mathbb{G}^{*l})^2 \mid \exists s \in \mathbb{Z}_l^*, pk_1 = pk_2{}^s\}$, 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. NGS then allows anyone to transform mpk into a "nickname", i.e., a different representative nk of the same class $[mpk]_\mathcal{R}$ without access to 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 trapdoor, while the opener can identify the member hidden behind a nickname, by iterating over all 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 3.3).

Definition 3.1 (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 elements of the NGS scheme. We briefly 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. A user can also choose not to join the group but still to interact with its members (for instance, verify that a message is signed with an existing nickname).*
- *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.*

Most of the times, 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 4.

Definition 3.2 (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 3.3 (NGS Global Variables) *NGS is build on the following set of global variables.*

- *$DS = \{KeyGen, Sig, Vf\}$ is a digital signature scheme that will be used in the implementation.*

- **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 3.4 (NGS Scheme) *A* nickname 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.*

- *$IKg : 1^\lambda \rightarrow (isk, ipk)$ is the key generation algorithm that takes a security parameter λ and outputs an issuer secret/public keys (isk, ipk).*

- *$OKg : 1^\lambda \rightarrow (osk, opk)$ is the key generation algorithm that takes a security parameter λ and outputs an opener's secret/public keys (osk, opk).*

- *$UKg : 1^\lambda \rightarrow (usk, upk)$ is the user key generation algorithm that produces appropriate secret/public keys (usk, upk).*

- *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.*

- $Nick$: $mpk \to nk$ *is the nickname generation algorithm that takes a master public key* mpk *and creates a nickname* nk *belonging to* $[mpk]_{\mathcal{R}}$.

- $Trace$: $(\tau, nk) \to b$ *takes as input the trapdoor* τ *for some equivalence class* $[mpk]_{\mathcal{R}}$ *and a nick* 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* 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 privacy and other security properties that NGS provides (see Section 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, in which case the opener will be able 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 its secret/public keys (usk, upk); it is assumed that upk is certified by a certification authority CA, and stored in **upk**. Then, to effectively 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. Meanwhile

(2), the issuer must run the Iss function to verify the correctness of this join request, in particular with respect to the user's public key, upk, and the opener public key, opk. 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, while its encrypted version is stored in **reg**, to allow the opener to identify his nicknames in the future (see Figure **??**).

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 The 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 (see Figure **??**).

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 The verifier checks that the signer controls nk by verifying the signature σ (see Figure **??**).

Opening The opener can, at any time, identify which user controls a nickname nk by testing all the users encrypted trapdoors stored in **reg** (see Figure **??**). 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 Implementation

We describe here, in pseudo code, a possible algorithmic implementation for the NGS variables and functions. It is an extension of the implementation, or "construction", introduced in [15], something which will help for proving the NGS security properties introduced in Section 4.

3.3.1 Types and variables

We define here our implementation choices for the NGS types and variables.

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. $\mathbb{G}_2$, part of a given pairing $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, with $\mathbb{G}_1 = \mathbb{Z}_p$.

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

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 $H : \mathbb{G}_1 \to \mathbb{G}_2$ be a hash function.

3.3.2 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 $DS.KeyGen(1^\lambda)$, while updating **upk**.
- $Join : (usk, opk) \to (msk, \tau, reqU)$.
  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 $\hat{S} = \hat{g}^s$ 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 $reqU = (f, w, \tau', \pi_J, \sigma_{DS})$. 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 in a previous or current joining session;

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 ().

- $Nick$: $mpk \rightarrow nk$. Let $(u, v, w) = mpk$. Then, select $r \leftarrow_{\$} \mathbb{Z}_p$. Let $nk = (u^r, v^r, w^r)$. Return nk.

- $Trace$: $(\tau, nk) \rightarrow b$. Let $(u, v, w) = nk$ and $b = (e(u, \tau) = e(w, \hat{g}))$. Return b.

- $Sign : (nk, msk, m) \rightarrow \sigma$. Let $(u, v, w) = nk$ and $\alpha = msk$. Compute and return $\sigma = SPK_S\{(\alpha) : w = u^\alpha\}(m)$.

- GVf : $(ipk, nk) \rightarrow b$. 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 : $(nk, m, \sigma) \rightarrow b$. Let $(u, v, w) = nk$ and return $b = true$ if the signature σ is valid with respect to (u, w) and m, $false$ otherwise.

- $Open : (osk, nk) \rightarrow (i, \Pi)$ 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 decrypt the user trapdoor τ by computing $\tau = \hat{f}' \cdot \hat{S}^{-z}$;
     (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).

- $Judge : (nk, i, ipk, \Pi) \rightarrow b$. 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.

4 Security model

As stated in the introduction, GS has been studied for several decades and formalized [5][4]. It therefore constitutes a solid basis for NGS to build its security model on [5], with the necessary modifications that we highlight here.

We split the group verification function of GS into the NGS GVf and UVf functions for better modularity. A forgery in NGS consists of a nickname nk passing the GVf function, along with its corresponding signature σ passing the UVf function on nk. NGS non-frameability and traceability properties thus require an adversary $\mathcal{A}$ to output such a forgery. The NGS opening function takes nicknames only, not the additional signature σ on it as is done in DGS. Concretely, this means that when a nickname nk is opened, either it returns the identity of the user controlling it, or it fails. But in the latter case, the opener can be assured that nk will not come with a valid σ; otherwise it would break traceability. However, in the NGS anonymity experiment, $\mathcal{A}$ is restricted, with no access to a $Trace$ oracle, and the *GOpen* oracle requires an additional σ on nk.

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

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

In Section 4.1, we first present the oracles used in our security model. Then we present, in the subsequent subsections, the NGS correctness, non frameability, traceability, opening soundness, and selfless anonymity properties (see Table 1).

4.1 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 this oracle 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.

- Ch$_b(i_0, i_1, m)$: for two identities i_0 and i_1 and a message m chosen by $\mathcal{A}$, this oracle outputs a challenge signature σ on m under identity i_b, for a random bit $b \in \{0, 1\}$.

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.

4.2 Correctness

Definition 4.1 *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).*

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

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

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

$\mathbf{Ch}_b(i_0, i_1, m)$
  If $(i_0, \tau) \notin L_h \vee (i_1, \tau) \notin L_h$ for some $\tau$
    return $\bot$
  If $i_0 \in L_{sk} \vee i_1 \in L_{sk}$ return $\bot$
  If $\mathbf{mpk}[i_b]$ undefined return $\bot$
  $nk = Nick(\mathbf{mpk}[i_b])$
  $\sigma = Sign(nk, \mathbf{msk}[i_b], m)$
  $L_{ch} = \{(i_0, m, nk, \sigma), (i_1, m, nk, \sigma)\} \cup L_{ch}$
  return $(nk, \sigma)$
```

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

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

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

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

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

Figure 1: Oracles for the security model of NGS

```
$Exp^{Corr}_{NGS}(\lambda, i, m)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $(usk, upk) = UKg(1^\lambda, i)$
  $(msk, \tau, reqU) = Join(usk, opk)$
  $Iss(i, isk, reqU, opk)$
  $nk = Nick(mpk)$
  $\sigma = Sign(nk, msk, m)$
  $(i', \Pi) = Open(osk, nk)$
  return $\left( \begin{array}{l} (i = i') \wedge \\ GVf(ipk, nk) \ \wedge \ UVf(nk, m, \sigma) \wedge \\ Judge(nk, i, ipk, \Pi) \ \wedge \ Trace(\tau, nk) \end{array} \right)$
```

Figure 2: Correctness experiment for NGS

Claim 4.1 *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then the NGS implementation of Section 3.3 is correct.*

The proof can be easily derived, notably from the PKs completeness.

4.3 Traceability

Traceability ensures that no adversary can create a valid signature opening to a non-registered user, i.e., untraceable. In this experiment, the opener is partially corrupt, 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 the *AddU* oracle 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 signature from a user who did not join the group, causing the *Open* algorithm to fail to identify the user. 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 $\mathcal{A} = \mathcal{A}^{AddU,CrptU,SndToI,USK,RReg}$ as

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

Definition 4.2 *A NGS scheme is* traceable *if $Adv^{Trace}_{NGS,\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and λ.*

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

Figure 3: Traceability experiment for NGS

Claim 4.2 *Let all PKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then then NGS construction is traceable under the Modified GPS assumption and the simulation soundness of PK_J.*

The proof can be found in Section A.1.

4.4 Non-frameability

Non-frameability ensures that no adversary can create a signed message linked to a valid nickname that, when opened, points to an honest user who did not actually produce it. 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^*, along with a target identity i^* and a proof Π^* accepted by the $Judge$ algorithm.

We define the advantage in the non-frameability experiment described in Figure 4 for any polynomial time adversary $\mathcal{A}$ as:

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

Definition 4.3 *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.*

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

Figure 4: Non-frameability experiment for NGS

Claim 4.3 *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then NGS is non-frameable under the simulation-soundness of SPK_S, the EUF-CMA of the underlying digital signature scheme DS, and the SDL assumption.*

The proof can be found in Section A.2.

4.5 Optimal opening soundness

Optimal opening soundness guarantees that no adversary can produce a signature that can be opened to two 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 signature σ on a message m 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. We define the advantage in the optimal opening soundness experiment described in Figure 5 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 4.4 *A NGS scheme is* optimally opening sound *if $Adv^{OS}_{NGS,\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and λ.*

```
$Exp^{OS}_{NGS,\mathcal{A}}(\lambda)$
  $(isk, ipk) = IKg(1^\lambda); (osk, opk) = OKg(1^\lambda)$
  $(nk^*, i^*_0, \Pi^*_0, i^*_1, \Pi_1) = \mathcal{A}^{AddU,CrptU,SndToI,USK,RReg}(ipk, osk)$
  If the following conditions hold, 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
```

Figure 5: Optimal opening soundness experiment for NGS

Claim 4.4 *The NGS construction is optimally opening sound under the simulation soundness of SPK_0 and SPK_2.*

The proof can be found in Section A.3.

4.6 Selfless anonymity

Anonymity ensures that no adversary can identify a specific signer from a target group, given his nickname nk^* and a signature σ^*. In this experiment, the issuer is corrupt, i.e, the adversary $\mathcal{A}$ is given the issuer key, but not the opener's. 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 two Ch_b oracles (for each value, 0 or 1, of b), 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 Sig oracle on a nickname nk for user i and message m of his choice.

We define the advantage in the anonymity experiment described in Figure 6 for any polynomial time adversary $\mathcal{A} = \mathcal{A}^{SndToU,USK,WReg,Sig,Ch_b}$ as:

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

Definition 4.5 *A NGS scheme is* anonymous *if $Adv^{Anon}_{NGS,\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and any λ.*

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

Figure 6: Anonymity experiment for NGS

Claim 4.5 *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then NGS is selfless anonymous under the zero-knowledge of the SPKs, the SXDH assumption, and the non-frameability of NGS.*

The proof can be found in Section A.4.

5 Application: NickHat

In this section, we describe a financial application of our NGS construction. We built a permissioned system that supports auditing and provides anonymity for tokens on the Ethereum blockchain. Although our system could work with any type of Ethereum currency, we focus on tokens that respect the ERC-20 norm.

Roles in this system are close to the 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 norm) and sends it to the blockchain and an operator that manages the system.

5.1 Description

We describe the main operations of NickHat. Assume user i already holds v units of a deployed ERC-20 token.

Setup First, an issuer and supervisor are chosen and run the NGS IKg and Okg functions respectively to generate new keys. The operator deploys the verifier smart contract that embeds the NGS UVf and GVf functions for the chosen issuer, the key registry contract that adds master public keys and displays it, the NickHat contract with *deposit*, *transfer*, *withdraw* functions, and 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 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 encrypted trapdoor. Finally, the operator saves the new master public key by sending a transaction to the key registry contract that makes it publicly readable.

Deposit User i wants to transfer v tokens to group member j 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 group member j master public key mpk from key registry contract and computes a nickname nk_j with the NGS Nick algorithm. He then signs a transaction to deposit v tokens to nk_j and sends it to the *Deposit* function of the NickHat contract. This contract spends the ERC-20 token allowance and transfer user i tokens to itself, acting as an escrow. The NickHat balances variable is then updated by setting v tokens for the specified nickname nk_j and token address. Finally, the NickHat contract emits an Announcement event with the nickname nk_j.

Detection Group member j listens to Announcement events of the blockchain and calls the $Trace$ function to check if the new nicknames belongs to him.

Transfer Group member j can unlock nk_j and sends v of his escrowed tokens to group member k. He first gets member k master public key from the key registry contract and calls the NGS *Nick* function to obtain a nickname nk_k. Then, he prepares a request to send v tokens from nk_j to nk_k and signs it with the NGS *Sig* function on the request hash. A relayer is then handed the request 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 update the balances and emits an Announcement event.

Withdraw User k listens to Announcement events and learns that nk_k is his. To withdraw his tokens to an Ethereum address, he proceeds the following way A request is first prepared stating that the owner of nk_k sends tokens to the specified address and 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 verifies the signature by calling the *UVf* of the verifier contract and redirects the call to the NickHat *withdraw* function that substracts tokens from nk_k balance and request the ERC-20 token contract to transfer tokens from the NickHat address to the recipient address.

Audit The supervisor can inspect any nickname announced in blockchain events. He can retrieve the identity of the user behind a nickname by calling the first part of the NGS *Open* algorithm. In case of 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.

5.2 Extension

We additionally test financial usecases such as delivery vs payment where a bond is atomically exchanged with cash. As explained above, the NickHat contract already holds the tokens, and it can be sufficient to extend it with hash time lock contract (HTLC) functions. For better modularity, we deployed an HTLC capable of exchanging tokens held in NickHat (and traditional ERC-20 tokens). An allowance variable is added that records who authorizes which spender to dispose of a specified number of tokens. An *approve* function is also added to the NickHat contract that updates the allowance variable for the owner, the nickname that sends the transaction, the spender, an Ethereum address, for a certain token and its quantity.

We assume a member Alice owns a nickname nk_a with v_1 units of token 1 and member Bob owns nk_b with v_2 units of token 2. Alice prepares a request to lock her tokens with a chosen hash and time lock into the HTLC contract, signing it with NGS Sig. This request is then wrapped into a meta-transaction signed by a relayer. The forwarder contract receives the request and verifies the signature with UVf. It then redirects the request to the HTLC *lock* function that proceeds to withdraw the tokens from NickHat to this HTLC contract. Bob does the same for token 2 chosing the same hash lock.

Alice then signs with NGS Sig function the request unlocking Bob's contract with the secret pre-image of the hash. As usual, the forwarder receives the meta-transaction and redirects the call to the HTLC's *unlock* function. This function calls the *deposit* function of the NickHat contract to deposit $v2$ tokens 2 to nk_a. Bob, learning the secret, can then unlock Alice's token.

6 Related work

NGS shares similarities with traditional group signatures yet with key differences. First, the *Open* and *Judge* functions only take as argument nicknames, not signatures. Second, NGS introduces the $Nick$ and $Trace$ functions, enabling users to be able to track the transactions in which they are involved. Finally, the verification function is here split in two functions, *GVf* and *UVf*.

Compared to SFPK, NGS splits the user's key generation of SFPK into the two parts within the joining protocol between the user and issuer. The first part consists of the generation of a secret key and trapdoor by the user; then the issuer produces the flexible public key, by signing the user's committed key as message. Note though that the SFPK class-hiding property is strictly stronger than NGS selfless anonymity, as a NGS adversary is not given the secret key of the target user [8]. Their unforgeability property is, however, encompassed by NGS non-frameability and traceability properties.

They additionally proposed a group signature construction based on structure-preserving signature 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. This construction avoids the user's proof of knowledge which can be costly[16] but also requires two adaptations. Indeed, in order to sign a message, the member first randomizes his SFPK public key, then has to adapt the representation of SPS-EQ with the same randomizer. Therefore, it requires more computation but also 3 times larger signatures than the ROM's ones. Our construction based on [15], with random oracle, offers an efficient proof of knowledge for the user as it takes place in $\mathbb{G}_1$. It thus avoids the two randomizations during the signature, while maintaining efficiency of the proof of knowledge.

7 Future work

We discuss in this section possible extensions for the NGS scheme and implementation presented above.

Stronger anonymity Our anonymity property protects nicknames that have been correctly randomized. For example, if an adversary sets $r = 0$ in the $Nick$ function, the nickname is equal to the master public key and anyone can therefore identify the user by looking at the **mpk** table. Of course, this particularity can also be seen as an optional anonymity feature. A stronger anonymity property would protect the recipient by requiring a correct randomization proof in the GVf function, for example. We stress that this limitation, or feature, is also evident in other stealth address schemes such as [10], where $r = 0$ in the Diffie-Hellman key exchange can also be set by the sender, leading to a stealth address equal to its master public key.

Stronger traceability The adversary in the NGS traceability experiment has to output a message-signature-nickname tuple (m, nk, σ) that passes the GVf algorithm and gives $Open(nk, \sigma) = \perp$ and $UVf(m, nk, \sigma)$. This means that, if a nickname nk passes the first two conditions, the adversary cannot come up with a valid σ. A stronger traceability property would not ask the adversary to provide the signature σ. In other words, the adversary cannot find a nickname nk such that $Open(osk, nk) \wedge GVf(ipk, nk)$.

Batch Processing Our NGS implementation, based on [15], inherits its so-called "batching property", which can be used for improving the performance of the GVf function. Consider a new verification function $BGVf : (\{nk_i\}_{i=1}^n) \rightarrow b$ that operates in batch mode, n nicknames nk_i at a time. Let $(u_i, v_i, w_i) = nk_i$ be the i^{th} nickname and q, a small prime number. This verifier would proceed as follows (we use the notations introduced into GVf).

1. Sample $e_1, .., e_n \leftarrow_{\$} \{0, 1\}^q$ and set $\{\tilde{u}_i, \tilde{v}_i, \tilde{w}_i\}_{i=1}^n = \{u_i^{e_i}, v_i^{e_i}, w_i^{e_i}\}_{i=1}^n$.
2. Check if $e(\prod_{i=1}^n \tilde{v}_i, \hat{g}) = e(\prod_{i=1}^n \tilde{u}_i, \hat{X}) \cdot e(\prod_{i=1}^n \tilde{w}_i, \hat{Y})$.
3. Return $true$ if the pairing check succeeds, and $false$, otherwise.

This reduces $3n$ pairing operations to 3, thanks to the small exponent test [3].

A Appendix

A.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 $(i, cont) \in L_c$ (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 $AddU$ 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 $reg = (i, \tau', \rho, \sigma_{DS}) \in$ **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 $(i^*, \Pi^*) = \mathit{Open}(osk, nk)$ and $\mathit{Judge}(nk^*, ipk, i^*, \Pi^*)$ 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.

A.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 obtain 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 $\sigma_{DS} = \mathit{DS.Sig}(\mathbf{usk}[i^*], \rho)$, τ' is the encryption $(\hat{g}^s, \hat{D}\hat{Z}^s)$ 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 $\neg Trace(\tau, nk)$, 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. Since 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 $\mathit{DS.Vf}(\mathbf{upk}[i^*], \rho^*, \sigma^*_{DS})$, 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.

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

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

with $Adv_{DS,\mathcal{A}}^{EUF}$, the advantage of the adversary in breaking the unforgeability of the DS signature, $Adv_{PK_O,\mathcal{A}}^{SS}$, 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.

A.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 *AddU* 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.

A.4 Selfless anonymity proof

For this proof, we chain experiments from G_0, the original experiment $Exp_{GNS,\mathcal{A}}^{Anon-0}$, 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 $Exp_{GNS,\mathcal{A}}^{Anon-1}$, 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 $Exp_{GNS,\mathcal{A}}^{Anon-0}$ 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 A.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 A.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 A.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 A.4.

Lemma A.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: $L_h = L_h \cup \{(i, \hat{g}^\alpha)\}$.

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 A.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.

- 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 A.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 A.1, since G_5 is indistinguishable from G_4, but for user i_1.

Lemma A.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 $\mathbf{msk}[i_1] = \alpha_1$.
- 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.

References

[1] Digital signature standard (dss), fips 186-5, 2023.

[2] Backes, M., Hanzlik, L., Kluczniak, K., and Schneider, J. Signatures with flexible public key: Introducing equivalence classes for public keys. In *International Conference on the Theory and Application of Cryptology and Information Security* (2018), Springer, pp. 405–434.

[3] Bellare, M., Garay, J. A., and Rabin, T. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology—EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31–June 4, 1998 Proceedings 17* (1998), Springer, pp. 236–250.

[4] Bellare, M., Micciancio, D., and Warinschi, B. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22* (2003), Springer, pp. 614–629.

[5] Bellare, M., Shi, H., and Zhang, C. Foundations of group signatures: The case of dynamic groups. In *Cryptographers' track at the RSA conference* (2005), Springer, pp. 136–153.

[6] Bernhard, D., Pereira, O., and Warinschi, B. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18* (2012), Springer, pp. 626–643.

[7] Bichsel, P., Camenisch, J., Neven, G., Smart, N., and Warinschi, B. Get shorty via group signatures without encryption. In *Security and Cryptography for Networks - SCN 2010* (Germany, 2010), vol. 6280, Springer Berlin Heidelberg, pp. 381–398. Other page information: 381-398 Conference Proceedings/Title of Journal: Security and Cryptography for Networks - SCN 2010 Other identifier: 2001254.

[8] Boneh, D., and Shacham, H. Group signatures with verifier-local revocation. In *Proceedings of the 11th ACM conference on Computer and communications security* (2004), pp. 168–177.

[9] Chaum, D., and Van Heyst, E. Group signatures. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10* (1991), Springer, pp. 257–265.

[10] Courtois, N. T., and Mercer, R. Stealth address and key management techniques in blockchain systems. In *ICISSP 2017-Proceedings of the 3rd International Conference on Information Systems Security and Privacy* (2017), pp. 559–566.

[11] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86* (Berlin, Heidelberg, 1987), A. M. Odlyzko, Ed., Springer Berlin Heidelberg, pp. 186–194.

[12] Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., and Simkin, M. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *Public-Key Cryptography–PKC 2016: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I* (2016), Springer, pp. 301–330.

[13] Goldwasser, S., Micali, S., and Rivest, R. L. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing 17*, 2 (1988), 281–308.

[14] Groth, J. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Advances in Cryptology–ASIACRYPT 2006: 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006. Proceedings 12* (2006), Springer, pp. 444–459.

[15] Kim, H., Lee, Y., Abdalla, M., and Park, J. H. Practical dynamic group signature with efficient concurrent joins and batch verifications. *Journal of information security and applications 63* (2021), 103003.

[16] Pointcheval, D., and Sanders, O. Short randomizable signatures. In *Topics in Cryptology - CT-RSA 2016* (Cham, 2016), K. Sako, Ed., Springer International Publishing, pp. 111–126.

[17] Schnorr, C.-P. Efficient signature generation by smart cards. *Journal of cryptology 4* (1991), 161–174.

[18] Shoup, V. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16* (1997), Springer, pp. 256–266.