Practical Dynamic Group Signature with Efficient Concurrent Joins and Batch Verifications

Hyoseung Kim* Youngkyung Lee† Michel Abdalla‡ Jong Hwan Park§

Abstract

Dynamic group signatures ($\mathcal{DGS}$) enable a user to generate a signature on behalf of a group of users, allowing a prospective user to join via an appropriate join protocol. A natural security requirement in the dynamic setting is to permit an adversary to concurrently perform join protocol executions. To date, most of $\mathcal{DGS}$ schemes do not provide the efficient concurrent join protocols in their security analysis, because of the need to use knowledge extractors. Also, $\mathcal{DGS}$ schemes have to provide efficient batch verifications for practical applications such as Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication, where a large number of group signatures should be verified in a very short time. In this paper, we propose a new practical $\mathcal{DGS}$ scheme that supports not only efficient concurrent joins but also batch verifications. The concurrent security is proven by showing that our join protocols are simulated without any knowledge extractor in security analysis. To do this, we introduce a modified Pointcheval-Sanders (PS) problem that can guarantee efficiently checking equality of discrete logarithms. In terms of efficiency, when considering a type-3 pairing, our $\mathcal{DGS}$ scheme has the advantages that the signature generation and verification are faster and especially our batch verification is at least 7 times faster in case of verifying 100 signatures, compared to other comparable pairing-based $\mathcal{DGS}$ schemes in the literature.

Keywords: Batch verification, Concurrent join, Dynamic group signature, Pointcheval-Sanders.

1 Introduction

A group signature [22] is a useful primitive that allows a user to anonymously sign a message on behalf of a group of users. In a group signature setting, a group manager (called an issuer) issues a group signing key to each user, and a group signature produced by a user is verified under a group public key associated with the group. A valid group signature preserves the signer's anonymity, but if necessary another group manager (called an opener) is able to identify the signer who produced a signature in question, breaking the anonymity. In general, group signatures are classified according to whether the group is static or dynamic. If the group is static (as in [5]), the issuer generates a group signing key for each user and distribute it to each user in the group, and thus a static group signature scheme can be used only in an environment that fully trusts the issuer. On the other hand, if the group is (partially) dynamic (as in [6, 36]), a prospective user chooses its own secret key with which the user joins the group by performing a join protocol with the issuer. Although there is a fully dynamic group setting where users can join and leave the group (as

*Korea University, Seoul, Korea. Email: hyoseung_kim@korea.ac.kr.
†Korea University, Seoul, Korea. Email: dudrudve@korea.ac.kr.
‡DIENS, École normale supérieure, CNRS, PSL University, and INRIA, Paris, France. Email: michel.abdalla@ens.fr.
§Sangmyung University, Seoul, Korea. Email: jhpark@smu.ac.kr.

in [13]), dynamic group signatures ($\mathcal{DGS}$) can indeed be transformed easily into other privacy-preserving cryptographic primitives (e.g., direct anonymous attestations [15] and enhanced privacy ID [16]). Hence, it is hard to overstate the importance of $\mathcal{DGS}$.

Motivation. The basic security requirements for a $\mathcal{DGS}$ scheme have been formalized by the work of Bellare, Shi, and Zhang [6] (and also by the independent work of Kiayias and Yung [36]), where they define three security notions called anonymity, non-frameability, and traceability. Among the three notions, the security of the join protocol executions in a dynamic group is related to the traceability, where malicious users interact with the issuer to obtain group signing keys. The goal of malicious users is then to create a new group signature that cannot be traced back to the signer who produced the signature in question. In the BSZ model, the traceability is defined in considering concurrent join protocols where an adversary is allowed to schedule all message delivery in any number of concurrent join sessions [6]. The property of concurrent join is a very attractive since many users can join at the same time, which could not be avoided in many practical applications. However, many of $\mathcal{DGS}$ schemes [10, 42] have proved the traceability only by allowing join protocol executions in a sequential order, meaning that more realistic joining processes, such as performing multiple sessions or interleaving separate sessions, are not permitted in their security analysis of the traceability. The reason of such incompleteness is that the previous $\mathcal{DGS}$ schemes require the so-called *knowledge extractor*. The knowledge extractor is a polynomial time algorithm that enables to extract a witness of a proof generated by a Fiat-Shamir transform. In fact, a personal secret key chosen by a malicious user needs to be extracted during their traceability proofs.

Moreover, what is worse is that all secret keys have to be extracted using the knowledge extractor, every time malicious users join via the sequential join protocols. In particular, when the traceability proof needs to *rewind* the adversary and redefine the outputs of the random oracles to extract the secret key,[1] it is known that those n sequential extractions end up with an exponential running time of 2^n [7]. Because of the exponential blowup, the number n of joining users should be sometimes limited to be a logarithmic size of $O(\log n)$, thereby allowing a very small number of users (e.g., $n = 80$) to join a $\mathcal{DGS}$ scheme. To overcome the drawback from the rewinding problem, there have been several approaches called straight-line extraction techniques, including online extractors [29] and adaptive proofs of knowledge [7]. Unfortunately, those straight-line extraction approaches result in fairly inefficient $\mathcal{DGS}$ schemes. For example, Bootel et al. [13] used straight-line extractable proofs of knowledge as well as accountable ring signatures in order to construct a fully $\mathcal{DGS}$ scheme. However, their group signature scheme yields public parameters of linear size and signatures of logarithmic size in the number of group members. In particular, although Fiat-Shamir transformed Sigma protocols have been useful for many $\mathcal{DGS}$ constructions, [8] showed that they cannot have straight-line extractors. Also, the existing $\mathcal{DGS}$ construction [24] was based on an Omega protocol [30] in which the Paillier encryption [41] is used as an extractable commitment in common reference string (CRS) model. Though such a CRS model allows to remove knowledge extractor in a more efficient way, it is required in reality to establish an additional *highly-trusted* setup for CRS (distinct from the $\mathcal{DGS}$ setup) which publishes the CRS but does not have access to an extraction trapdoor (i.e., decryption key for the Paillier encryption). Naturally, a better solution than those straight-line extractors and CRS models is to eliminate the need of knowledge extractors when dealing with the join protocols. Recently, Derler and Slamanig [25] proposed a $\mathcal{DGS}$ scheme where the traceability proof does not rely on a knowledge extractor, meaning that their join protocols are concurrently secure.

However, despite achieving concurrent security, those $\mathcal{DGS}$ schemes [24] and [25] are still not sufficient for practical applications where efficient batch verifications are required, such as privacy-preserving authentication for Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication [26, 40, 46]

[1] Sometimes, it is called 'freeze-extract-resume' approach. [7]

and compact e-cash [20]. Indeed, according to the dedicated short range communication (DSRC) specification [43] (based on IEEE 802.11p [34]), each vehicle has to broadcast its status message every 300 ms. In such an environment, a vehicle may periodically receive a large number of group signatures and thus verifying such group signatures in a very short time becomes an inevitable challenge. Until now, almost all of $\mathcal{DGS}$ schemes are constructed based on bilinear maps, i.e., parings denoted by $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and when taking a type-3 pairing like BN curves [3] the efficiency for batch verifications is largely determined by the number of expensive operations such as exponentiations over $\{\mathbb{G}_2, \mathbb{G}_T\}$ and pairings. To the best of our knowledge, almost all previous pairing-based $\mathcal{DGS}$ schemes in the literature suffer from those expensive operations that increase linearly with the number n of signatures. This can be a critical reason why it is difficult to apply group signatures actively in the V2V and V2I communication.

Our Contribution. The goal of this paper is to propose a new practical $\mathcal{DGS}$ scheme which supports not only efficient concurrent joins but also pairing-batch verifications. Our $\mathcal{DGS}$ scheme follows the *sign-randomize-prove* (SRP) paradigm [10], where a randomizable signature is used as a building block. We adopt the randomizable signature suggested by Pointcheval and Sanders [42], but suggest a novel and efficient way of producing a group signature. In terms of efficiency, our $\mathcal{DGS}$ scheme features that all exponentiations (but pairings) during signature generation and verification are all computed over $\mathbb{G}_1$ with relatively small-sized group elements, and a batch verification for n signatures requires only the constant 3 pairings, regardless of the number n, and $2n$ exponentiations over $\mathbb{G}_1$. When considering a type-3 BN curve [3], for instance, the signature generation and verification are faster than other pairing-based $\mathcal{DGS}$ schemes in the literature. Moreover, our batch verification is at least 5 times faster for $n = 20$, and at least 7 times faster for $n = 100$, compared to the recent work [25] and the efficiently batch-verifiable variant [27] of Boneh, Boyen, and Shacham [11]. The opening of our $\mathcal{DGS}$ requires $O(n)$ search operations in the number n of group members, which inherently comes from the SRP paradigm.

In terms of security, our $\mathcal{DGS}$ scheme provides concurrently secure joins by showing that our join protocol executions are simulated without knowledge extractor in the traceability proof. We achieve the concurrent security under a modified Pointcheval-Sanders (PS) problem [42]. Originally, the PS problem is equipped with a signing oracle that takes $(u,m) \in \mathbb{G}_1 \times \mathbb{Z}_p$ as input and outputs $v = u^{x+my} \in \mathbb{G}_1$, where p is a group order of $\mathbb{G}_1$. Requesting the exponent m to the input straightforwardly leads to the need of knowledge extractor to obtain m. Importantly, changing an exponent m to a group element u^m is the key to eliminate the knowledge extractor in joining protocol executions of our traceability proof. Instead, along with the oracle input, the modified PS problem requires an adversary to additionally submit $(g, f = g^m)$, where g is a public element, in order to ensure that $\log_g f = \log_u w$. However, we notice that given an adversary's forgery $(u^*, w^* = (u^*)^{m^*}) \in \mathbb{G}_1^2$ in the traceability proof, the knowledge extractor is still needed to extract a message $m^* \in \mathbb{Z}_p$ (as a discrete logarithm) to solve the modified PS problem. In Section 8, we will prove that the modified PS problem holds in the generic group model [45].

1.1 Organization

We begin by describing preliminaries in Section 2. In Section 3, we present complexity assumptions including a new assumption that is a variant of the PS assumption. We give definitions of group signatures in Section 4. We then propose a practical dynamic group signature scheme being able to support efficient batch verifications in Section 5, and prove it using the new assumption in Section 6. We compare the proposed scheme with existing group signature schemes in Section 7. Section 8 provides proofs of assumptions that we introduced. We finally conclude the result of this paper in Section 9.

2 Preliminaries

We will denote by $\mathbb{Z}_p$ the set $\{0,\ldots,p-1\}$, where p is a prime. A function $f:\mathcal{R}\rightarrow\mathcal{R}$ is called *negligible* if for any $d>0$, $|f(k)|<1/k^d$ for sufficiently large k. We write $\mathsf{P}(\mathsf{A}(str_A)\leftrightarrow\mathsf{B}(str_B))\rightarrow(out_A;out_B)$ to indicate that the protocol P is composed of the interactive algorithms A and B, where the algorithms respectively take as inputs str_A and str_B and then output results out_A and out_B. As usual, we will use λ to denote a security parameter. Throughout this paper, all group operations are performed modulo p unless otherwise stated for the sake of clarity.

2.1 Bilinear Maps

Let a bilinear group be denoted by $\mathcal{G}=(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,g,\hat{g})$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are groups of prime order p, e is a bilinear map such that $e:\mathbb{G}_1\times\mathbb{G}_2\rightarrow\mathbb{G}_T$, g and $\hat{g}$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. A bilinear map (i.e., pairing) e satisfies the following properties for any g and $\hat{g}$; (1) *bilinearity*: $e(g^x,\hat{g}^y)=e(g,\hat{g})^{xy}$ holds, (2) *non-degeneracy*: $e(g,\hat{g})$ is a generator of $\mathbb{G}_T$, and (3) *efficiency*: computing $e(\cdot,\cdot)$ is efficient. In this paper, we will make use of a type-3 pairing for efficient operations in $\mathbb{G}_1$, such as BN curves [3].

2.2 Digital Signatures

A digital signature scheme consists of three algorithms; (1) $\mathsf{DSKg}(1^\lambda)\rightarrow(sk,vk)$: a key generation algorithm, which outputs a signing key sk and a public key vk under a security parameter λ, (2) $\mathsf{DSSig}(sk,m)\rightarrow\sigma_{DS}$: a signing algorithm, which takes as inputs a singing key sk and a message m to be signed and then outputs a signature σ_{DS}, and (3) $\mathsf{DSVf}(vk,m',\sigma)\rightarrow\{0,1\}$: a verifying algorithm, which takes a verifying key vk, a message m', and a signature σ_{DS} and then returns 1 if (m,σ) is valid under vk; otherwise 0. We consider existential unforgeability under chosen message attacks (EUF-CMA) [31] as the security of a digital signature scheme.

2.3 Proof Protocols

A signature proof of knowledge (SPK) is a non-interactive zero-knowledge (NIZK) proof using the Fiat–Shamir transformation [28] in the random oracle model. An SPK π on a message m is formally denoted in [21], and is described in a generic way in [10] for proving a pre-image of a group homomorphism $\phi:\mathbb{G}\rightarrow\mathbb{G}'$ with two groups of prime order p as below:

$$\pi=SPK\{(x):y=\phi(x)\}(m),\ \text{where } m\in\{0,1\}^*\cup\{\perp\}.$$

Without loss of generality, we use the additive notation for $\mathbb{G}$ and the multiplicative notion for $\mathbb{G}'$. Let $h:\{0,1\}^*\rightarrow\mathbb{G}$ be a hash function. The proof π is set to the tuple (c,s) such that $c=h(\phi,y,\phi(\mathsf{rnd}),m)$ and $s=\mathsf{rnd}-c\cdot x\in\mathbb{G}$ where $\mathsf{rnd}\in\mathbb{G}$ is chosen at random. If π is valid then $c=h(\phi,y,y^c\phi(s),m)$ holds. The syntax languages of this SPK can be connected by the AND operator (i.e., $\wedge$). We use $y_1=\phi_1(x_1)\wedge y_2=\phi_2(x_2)$ to denote $\phi(x_1,x_2)=(\phi_1(x_1),\phi_2(x_2))$ or use $y_1=\phi_1(x)\wedge y_2=\phi_2(x)$ to denote $\phi(x)=(\phi_1(x),\phi_2(x))$.

The properties of SPKs (i.e., NIZKs) have been defined in [9, 32], which can be informally described with notions of SPK as follows:

- *Completeness.* A signature generated by an honest signer should be verified successfully.

- *Zero-knowledge (ZK).* There exists a zero-knowledge simulator $\mathcal{S}$, not knowing witness, which is able to simulate a valid proof (i.e., a signature of SPK) indistinguishable from a real one.

- *Simulation soundness (SS).* A cheating signer with no witness is unable to generate a new proof for a false statement even if receiving simulated proofs.

- *Simulation-sound extractability (SE).* There exists a knowledge extractor $\mathcal{E}$ to extract a correct witness from a valid proof generated by a cheating signer, who receives simulated proofs for arbitrary statements.

Definition 2.1. *An SPK is a simulation-sound extractable NIZK if it satisfies completeness, zero-knowledge, and simulation-sound extractability.*

3 Complexity Assumptions

We introduce complexity assumptions that hold in type-3 pairing groups. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \hat{g})$ be a type-3 pairing group.

3.1 Variants of the PS assumption

PS Assumption [42]. Given a tuple $(\hat{g}^x, \hat{g}^y)$ and an oracle $\mathcal{O}^{PS}(m) \to (u, u^{x+ym}) \in \mathbb{G}_1^2$, where $m \in \mathbb{Z}_p$ and $u \in \mathbb{G}_1$ is uniformly distributed, the PS problem is to find a tuple $(u', v', m) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that $u' \neq 1$, $v' = u'^{x+ym}$, and m has not been queried.

Generalized PS Assumption. Given a tuple $(\hat{g}^x, \hat{g}^y)$ and two oracles such that; $\mathcal{O}_0^{GPS}(\cdot) \to u$, where $u \in \mathbb{G}_1$ is uniformly distributed, $\mathcal{O}_1^{GPS}(u, m) \to v$, where $u \in \mathbb{G}_1$, $m \in \mathbb{Z}_p$, and $v = u^{x+ym} \in \mathbb{G}_1$ as output only if u was provided by $\mathcal{O}_0^{GPS}$ and was not queried before. The generalized PS (GPS) problem is to find a tuple $(u', v', m) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that $u' \neq 1$, $v' = u'^{x+ym}$, and m has not been queried.

Modified GPS Assumption. Given a tuple $(\hat{g}^x, \hat{g}^y)$ and the following three oracles;

- $\mathcal{O}_0^{MGPS}(\cdot) \to u$, where $u \in \mathbb{G}_1$ is random.
- $\mathcal{O}_1^{MGPS}(g, u, f, w) \to v$, where $(g, u, f, w) \in \mathbb{G}_1^4$ as input and $v = u^x w^y \in \mathbb{G}_1$. The output v is returned only if (1) u was not queried to this oracle before, (2) u was previously appeared in $\mathcal{O}_0^{MGPS}$ as output, and (3) $\log_g f = \log_u w$.

The modified GPS problem is to find a tuple $(u', v', m) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ such that $u' \neq 1$ and $v' = u'^{x+ym}$ for a new m. That is, for any pair (u, w) queried to $\mathcal{O}_1^{MGPS}$ the equality $w = u^m$ does not hold.

The GPS assumption is almost identical to the generalized LRSW [17] previously obtained by expanding the LRSW [38]. Here, the GPS assumption is defined as one step towards reaching the next modified GPS assumption, and thus is not used for our security analysis. Nevertheless, it might be useful for some constructions where the base u should be randomly chosen before determining a message m.

The modified GPS assumption has the following difference, which is that (u, w) (and others) are given to $\mathcal{O}_1^{MGPS}$ as input and the corresponding output is computed as $v = u^x w^y$. The GPS assumption requires $(u, m) \in \mathbb{G}_1 \times \mathbb{Z}_p$ as input to generate $v = u^{x+ym}$, but eventually requesting m as the *exponent* causes a knowledge extractor to be required in security analysis. That is, the change from $m \in \mathbb{Z}_p$ to $w \in \mathbb{G}_1$ enables us to remove a knowledge extractor. Instead, the modified GPS assumption additionally requires an adversary to

provide a pair $(g, f = g^m)$ to ensure that 'it really knows the exponent m'. Specifically, given a random base u (as the output of $\mathcal{O}_0^{MGPS}$), it is infeasible to compute (g, u, f, w) (as the input of $\mathcal{O}_1^{MGPS}$) such that $f = g^m$ and $w = u^m$ without knowing m. This is assured basically by the knowledge of exponent (KEA) assumption[2] [23] defined without knowledge extractors. In Section 8, we will prove that those two assumptions (defined here) hold in the generic group model [45].

3.2 Other complexity assumptions

Symmetric External Diffie–Hellman (XDH) Assumption [11]. Given a tuple (g, g^a, g^b, T), where a, b are random in $\mathbb{Z}_p$, the XDH problem in $\mathbb{G}_1$ ($\mathrm{XDH}_{\mathbb{G}_1}$) is to determine whether T is g^{ab} or random in $\mathbb{G}_1$. The $\mathrm{XDH}_{\mathbb{G}_1}$ assumption holds if the advantage of solving this problem is negligible in λ. Similarly, the XDH assumption in $\mathbb{G}_2$ ($\mathrm{XDH}_{\mathbb{G}_2}$) holds if the advantage of solving such a problem is negligible in $\mathbb{G}_2$. The symmetric XDH (SXDH) assumption holds if the XDH assumptions hold in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

Symmetric Discrete Logarithm (SDL) Assumption [10]. Given a tuple $(g, g^d, \hat{g}, \hat{g}^d)$, where d is random in $\mathbb{Z}_p$, the SDL problem is to find d. The SDL assumption holds if the advantage of solving the SDL problem is negligible in λ.

4 Definitions

Following [6], we begin with the syntax for $\mathcal{DGS}$ and then provide formal security models of $\mathcal{DGS}$.

4.1 Syntax

$\mathsf{GKg}(1^\lambda) \to (gpk, ik, ok)$. The group-key generation is run by the trusted party. Given a security parameter λ, this algorithm outputs the issuer's public/secret key pair (ipk, ik) and the opener's public/secret key pair (opk, ok). A group public key is published by $gpk = (ipk, opk)$.

$\mathsf{UKg}(1^\lambda, i) \to (upk_i, usk_i)$. A 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).

$\mathsf{GJoin}(\mathsf{Join}(i, usk_i, gpk) \leftrightarrow \mathsf{Iss}(i, upk_i, ik)) \to (gsk_i, reg_i)$. A user i executes the GJoin protocol with the issuer. The algorithm Join (run by the user i) takes the user secret key usk_i and the group public key gpk. The algorithm Iss (run by the issuer) takes the user public key upk_i and its secret key ik. At the end of the protocol, the user obtains a group signing key gsk_i and the issuer adds the registration information reg_i on the user i into the registration list **reg**. This list will be shared with the opener, so that the opener has access to read **reg**.

$\mathsf{GSig}(gsk_i, m) \to \sigma$. A user i having its group signing key gsk_i invokes the signing algorithm to produce a signature σ on a message $m \in \{0,1\}^*$ on behalf of the group.

$\mathsf{GVf}(gpk, m, \sigma) \to \{0,1\}$. Anyone can invoke the verification algorithm to verify a group signature. The algorithm takes the group public key gpk, a message m, and a signature σ and then outputs 1 if σ is valid on m; otherwise, it outputs 0.

[2]This assumption was introduced by Damgård [23]. The definition is that for an adversary that takes a tuple $(g, u = g^\varsigma)$, it is infeasible to compute $(f, w = f^\varsigma)$ if it does not know m such that $g^m = f$.

$\mathsf{GOpen}(ok, m, \sigma, \mathbf{reg}) \to (i, \Pi)$ or $\perp$. The opener invokes the opening algorithm to open an identity i of a user who produced a given signature. The algorithm takes the opening key ok, a message m, a valid signature σ, and the registration list **reg**. If such a user i exists, the algorithm outputs the identity i and a proof Π claiming that the user i produced σ on m; otherwise, it outputs $\perp$.

$\mathsf{GJudge}(m, \sigma, gpk, i, upk_i, \Pi) \to \{0,1\}$. Anyone invokes the judging algorithm to verify the opener's opening. The algorithm takes a message m, a valid signature σ on m, the group public key gpk, an identity i along with its public key upk_i, and a proof Π generated while opening. It outputs 1 if Π is valid; otherwise, it outputs 0.

Definition 4.1. *A $\mathcal{DGS}$ scheme is correct if the following three conditions hold:*

$$\mathsf{Pr}\left[\begin{array}{c} \mathsf{GVf}(gpk, m, \sigma) = 1 \wedge \\ \mathsf{GJudge}(m, \sigma, gpk, i, upk_i, \Pi) = 1 \\ \wedge i = j \end{array} : \begin{array}{c} (gpk, ik, ok) \leftarrow \mathsf{GKg}(1^\lambda), \\ (upk_i, usk_i) \leftarrow \mathsf{UKg}(1^\lambda), \\ (gsk_i, reg_i) \leftarrow \mathsf{GJoin}\big(\mathsf{Join}(i, usk_i, gpk) \\ \leftrightarrow \mathsf{Iss}(i, upk_i, ik)\big), \\ reg_i \in \mathbf{reg},\ \sigma \leftarrow \mathsf{GSig}(gsk_i, m), \\ (j, \Pi) \leftarrow \mathsf{GOpen}(ok, m, \sigma, \mathbf{reg}) \end{array}\right] = 1.$$

4.2 Formal Security Models

We mainly refer to the BSZ model [6] that is considered as the strongest security model for $\mathcal{DGS}$ to date. Our security model is almost the same as the BSZ model, except that we provide a slightly weaker anonymity notion called the 'selfless anonymity' where signing key queries with respect to target users are disallowed. The relaxed anonymity allows us to obtain a much more efficient $\mathcal{DGS}$ scheme, especially in terms of pairing-batch verifications.

4.2.1 Oracle Description.

To define security models in which the challenger $\mathcal{C}$ interacts with an adversary $\mathcal{A}$, we give the description of oracles used by $\mathcal{A}$.

$\mathsf{AddU}(i)$. $\mathcal{A}$ uses this oracle to add and then join an honest user i.

$\mathsf{CrptU}(i, upk)$. $\mathcal{A}$ uses this oracle to corrupt a user i and set the user public key to be upk of its choice.

$\mathsf{SndToI}(i)$. $\mathcal{A}$ (on behalf of a malicious user i) uses this oracle to join a malicious user i, by executing GJoin with an honest issuer. $\mathcal{A}$ does not need to follow the way scheduled in GJoin.

$\mathsf{SndToU}(i)$. $\mathcal{A}$ (on behalf of a malicious issuer) executes GJoin with an honest user i in order to join. $\mathcal{A}$ does not need to follow the way scheduled in Iss.

$\mathsf{USK}(i)$. $\mathcal{A}$ uses this oracle to get private information such as gsk_i and usk_i that an honest user i internally stored.

$\mathsf{RReg}(i)$. Using this oracle, $\mathcal{A}$ can read the entry for a user i stored in the registration list.

$\mathsf{WReg}(i, \rho)$. Using this oracle, $\mathcal{A}$ can write (or modify) the entry for a user i as ρ of its choice.

$\underline{\mathsf{AddU}}(i)$:
- $(upk_i, usk_i) \leftarrow \mathsf{UKg}(1^\lambda, i)$.
- $(gsk_i; reg_i) \leftarrow \mathsf{GJoin}(\mathsf{Join}(i, usk_i, gpk) \leftrightarrow \mathsf{Iss}(i, upk_i, ik))$.
- $\mathbf{reg} \leftarrow \mathbf{reg} \cup \{reg_i\}$.
- $\mathcal{L}_h \leftarrow \mathcal{L}_h \cup \{i\}$.
- Return upk_i.

$\underline{\mathsf{CrptU}}(i, upk)$:
- $upk_i \leftarrow upk$.
- $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{(i, \mathsf{cont})\}$.

$\underline{\mathsf{SndToI}}(i)$:
- If $(i, \mathsf{cont}) \notin \mathcal{L}_c$ then return $\perp$.
- $reg_i \leftarrow \mathsf{GJoin}(\mathcal{A} \leftrightarrow \mathsf{Iss}(i, upk_i, ik))$.
- $\mathbf{reg} \leftarrow \mathbf{reg} \cup \{reg_i\}$.
- $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{(i, \mathsf{accept})\}$.

$\underline{\mathsf{SndToU}}(i)$:
- $(upk_i, usk_i) \leftarrow \mathsf{UKg}(1^\lambda, i)$.
- $gsk_i \leftarrow \mathsf{GJoin}(\mathsf{Join}(i, usk_i, gpk) \leftrightarrow \mathcal{A})$.
- $\mathcal{L}_h \leftarrow \mathcal{L}_h \cup \{i\}$.

$\underline{\mathsf{USK}}(i)$:
- If $(i, *, *) \in \mathcal{L}_{ch}$ then return $\perp$.
- $\mathcal{L}_{sk} \leftarrow \mathcal{L}_{sk} \cup \{i\}$.
- Return (gsk_i, usk_i).

$\underline{\mathsf{RReg}}(i)$:
- Return reg_i.

$\underline{\mathsf{WReg}}(i, \rho)$:
- $reg_i \leftarrow \rho$.

$\underline{\mathsf{Sig}}(i, m)$:
- If $i \notin \mathcal{L}_h$, then return $\perp$.
- $\sigma \leftarrow \mathsf{GSig}(gsk_i, m)$.
- $\mathcal{L}_\sigma \leftarrow \mathcal{L}_\sigma \cup \{(i, m, \sigma)\}$
- Return σ.

$\underline{\mathsf{Open}}(m, \sigma)$:
- If $(*, m^*, \sigma^*) \in \mathcal{L}_{ch}$, then return $\perp$.
- Return $\mathsf{GOpen}(ok, m, \sigma, \mathbf{reg})$.

$\underline{\mathsf{CH}_b}(i_0^*, i_1^*, m^*)$:
- If $i_0^* \notin \mathcal{L}_h$ or $i_1^* \notin \mathcal{L}_h$ then return $\perp$.
- If $i_0^* \in \mathcal{L}_{sk}$ or $i_1^* \in \mathcal{L}_{sk}$ then return $\perp$.
- $\sigma^* \leftarrow \mathsf{GSig}(gsk_{i_b^*}, m^*)$.
- $\mathcal{L}_{ch} \leftarrow \{(i_0^*, m^*, \sigma^*), (i_1^*, m^*, \sigma^*)\}$.
- Return σ^*.

Figure 1: Oracles used by an adversary

$\mathsf{Sig}(i, m)$. $\mathcal{A}$ uses this oracle to obtain a signature on a message m under a group signing key gsk_i corresponding to an honest user i.

$\mathsf{CH}_b(i_0^*, i_1^*, m^*)$. For two i_0^*, i_1^* and a message m^* chosen by $\mathcal{A}$, this oracle outputs a challenge signature σ^* on m^* under i_b^* for a random bit $b \in \{0, 1\}$.

$\mathsf{Open}(m, \sigma)$. Using this oracle, $\mathcal{A}$ can obtain the result of the opening algorithm run by an honest opener, with respect to a message m and a signature σ. The challenge signature σ^* output by Ch_b is not allowed to query to this oracle.

These oracles are specifically described in code type in Figure 1. In addition, $\mathcal{C}$ maintains the following lists to control those oracles. These lists are set to be initially empty.

- $\mathcal{L}_h$: a list of honest users.

- $\mathcal{L}_c$: a list of a corrupted user and its 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.

- $\mathcal{L}_{sk}$: a list of identities queried to the user secret key oracle.

- $\mathcal{L}_\sigma$: a list of queried identity and message, and a signature in response to the signing oracle.

- $\mathcal{L}_{ch}$: a list of challenged message and identities, and a signature in response to the challenge oracle.

4.2.2 Security Models.

Figure 2 presents the security models for anonymity, non-frameability, and traceability.

Anonymity. Anonymity ensures that no adversary can identify the signer from the target group signature. This notion allows $\mathcal{A}$ to collude with a malicious issuer, but an opener should be essentially kept honest. Since the issuer is fully corrupted, $\mathcal{A}$ is initially given ik, the issuer's secret key. $\mathcal{A}$ thereby uses the SndToU oracle to allow an honest user to join, but does not need to use the CrptU oracle. $\mathcal{A}$ is also allowed to modify a registration list via the WReg oracle and use the Open oracle to obtain the opening result by the honest opener. $\mathcal{A}$ has access to the CH_b oracle for two challenge identities (i_0^*, i_1^*) and a message m^*. A challenge signature σ^* returned from $\mathsf{CH}_b(i_0^*, i_1^*, m^*)$ is given to $\mathcal{A}$. Of course, (m^*, σ^*) is not permitted to be queried to the Open oracle. Especially, regarding the USK oracle, $\mathcal{A}$ can use the oracle to obtain secret keys for all users except the target users with identities i_0^*, i_1^*, which is called the *selfless anonymity* [12, 19]. Obviously, the restriction makes the anonymity (defined here) weaker than that of the BSZ model [6], but the selfless anonymity is sufficient for realistic applications, when considering existing anonymity notions of anonymous identity-based encryption [1] and direct anonymous attestation [18]. $\mathcal{A}$ also uses the Sig oracle to obtain group signatures for any identity, including the target identities.

For any λ and any polynomial-time adversary $\mathcal{A}$, we define the advantage in the anonymity experiment in Figure 2 as

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

Definition 4.2. *A group signature scheme $\mathcal{DGS}$ is anonymous if $\mathbf{Adv}^{Anon}_{\mathcal{DGS},\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and any λ.*

Non-frameability. This notion means that no adversary can create a valid signature and a judge-accepted proof of an honest user who has not actually produced such a signature. In other words, this notion ensures that an honest user is not falsely accused of producing a certain group signature unless the user really did produce the signature. In this notion, the issuer and the opener are assumed to be all corrupted, so that $\mathcal{A}$ receives the issuer's secret key (ik) as well as the opener's secret key (ok). Thus, $\mathcal{A}$ is allowed to use the SndToU oracle, but does not need to use the CrptU, WReg, and Open oracles. Also, $\mathcal{A}$ has access to the USK oracle except for the target identity i^* ($\mathcal{A}$ outputs) and the Sig oracle except for the pair (i^*, m^*) ($\mathcal{A}$ outputs). The goal of $\mathcal{A}$ is then to create a valid forgery σ^* on a message m^*, and a target (opened) identity i^*, and a proof Π^* that are accepted by the GJudge algorithm.

For any λ and any polynomial-time adversary $\mathcal{A}$, we define the advantage in the non-frameability experiment in Figure 2 as

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

$\mathbf{Exp}^{Anon-b}_{\mathcal{DGS},\mathcal{A}}(\lambda)$:

- $(gpk, ik, ok) \leftarrow \mathsf{GKg}(1^\lambda)$.
- $b' \leftarrow \mathcal{A}^{\mathsf{SndToU},\ \mathsf{USK},\ \mathsf{WReg},\ \mathsf{Sig},\ \mathsf{Open},\mathsf{CH}_b}(gpk, ik)$.
- Return b'.

$\mathbf{Exp}^{Nf}_{\mathcal{DGS},\mathcal{A}}(\lambda)$:

- $(gpk, ik, ok) \leftarrow \mathsf{GKg}(1^\lambda)$.
- $(m^*, \sigma^*, i^*, \Pi^*) \leftarrow \mathcal{A}^{\mathsf{SndToU},\ \mathsf{USK},\ \mathsf{Sig}}(gpk, ik, ok)$.
- If the following conditions hold, then return 1.
  (1) $\mathsf{GVf}(gpk, m^*, \sigma^*) = 1$.
  (2) $i^* \in \mathcal{L}_h$ and $i^* \notin \mathcal{L}_{sk}$ and $(i^*, m^*, *) \notin \mathcal{L}_\sigma$.
  (3) $\mathsf{GJudge}(m^*, \sigma^*, gpk, i^*, upk^*, \Pi^*) = 1$.
- Return 0.

$\mathbf{Exp}^{Trace}_{\mathcal{DGS},\mathcal{A}}(\lambda)$:

- $(gpk, ik, ok) \leftarrow \mathsf{GKg}(1^\lambda)$.
- $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{AddU},\ \mathsf{CrptU},\ \mathsf{SndToI},\ \mathsf{USK},\ \mathsf{RReg}}(gpk, ok)$.
- If the following conditions hold, then return 1.
  (1) $\mathsf{GVf}(gpk, m^*, \sigma^*) = 1$.
  (2) When $(i^*, \Pi^*) \leftarrow \mathsf{GOpen}(ok, m^*, \sigma^*, \mathbf{reg})$,
  $i^* = \perp$ or $\mathsf{GJudge}(m^*, \sigma^*, gpk, i^*, upk^*, \Pi^*) = 0$.
- Return 0.

Figure 2: Security experiments for $\mathcal{DGS}$

Definition 4.3. *A group signature scheme $\mathcal{DGS}$ is non-frameable if $\mathbf{Adv}^{Nf}_{\mathcal{DGS},\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and any λ.*

Traceability. Traceability implies that no adversary can produce a valid signature that is eventually untraceable via the GOpen and GJudge algorithms. This notion holds when the issuer is honest and the opener is partially corrupt, where being partially corrupt means that $\mathcal{A}$ controls the opener but the opening executions must follow the prescribed program. $\mathcal{A}$ is allowed to call the AddU oracle to join an honest user, the CrptU and SndToI oracles to join a corrupt user, and eventually the USK oracle to obtain secrets for *all* honest users. Since secret keys of all users are available to $\mathcal{A}$, the Sig oracle is redundant and thus not

given. The only unknown information to $\mathcal{A}$ then becomes the issuer's secret key (ik), and thus via the RReg oracle $\mathcal{A}$ is able to read a registration list under the control of the honest issuer. Now, there are two ways that $\mathcal{A}$ succeeds. One is to generate a valid signature generated by an *unjoined* user, meaning that the GOpen algorithm fails to identify the origin of the forged signature. The other is to produce a valid signature that is generated by a *joined* (corrupted) user i^* but not traced back to the user i^* via the GOpen and GJudge algorithms. The latter should include the case where the GOpen algorithm outputs j^* although the forged signature is produced by the user i^*. However, by checking whether a user's secret key is uniquely registered during the join protocol executions, such a case of breaking the soundness of the GOpen algorithm can be easily removed. The latter thus includes the case of breaking the soundness of the GJudge algorithm, meaning that a proof Π^* by the GOpen algorithm is rejected by the GJudge algorithm.

For any λ and any polynomial-time adversary $\mathcal{A}$, we define the advantage in the traceability experiment in Figure 2 as

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

Definition 4.4. *A group signature scheme $\mathcal{DGS}$ is traceable if $\mathbf{Adv}^{Trace}_{\mathcal{DGS},\mathcal{A}}(\lambda)$ is negligible for any $\mathcal{A}$ and any λ.*

5 Our Dynamic Group Signature Scheme

5.1 Overview

Our $\mathcal{DGS}$ scheme follows the SRP paradigm, based on the PS signature [42] as an underlying randomizable signature scheme. The original PS signature on a message m is composed of $(u, v = u^{x+ym})$, where $u, v \in \mathbb{G}_1$ and x, y are signing key. Now, we transform it to add one more element such that $w = u^m \in \mathbb{G}_1$ into the signature, borrowing the idea from the DAA construction [18]. In the PS signature, such w is redundant because m and u are known to any verifier. However, in our group signature, m is randomly chosen by a joining user from an exponentially large space, and used as a signing key. Then, randomizing a signature (u, v, w) is still simply done by rasing a random exponent r to each group element. This change entails a slight increase in the signature size, but we can achieve significantly better computational efficiency as follows: when singing a real message $\tilde{m}$, after randomization, we use $(u, w = u^m)$ to produce a SPK proof π of knowledge of m. Importantly, all of these signing operations take place in $\mathbb{G}_1$ with relatively smaller-sized group elements. Given a group signature (u, v, w) and π, the verification is done by two steps; (1) check if $e(v, \hat{g}) \stackrel{?}{=} e(u, \hat{X})e(w, \hat{Y})$, using public parameters $\hat{g}, \hat{X} = \hat{g}^x, \hat{Y} = \hat{g}^y \in \mathbb{G}_2$, and (2) check if π is valid (along with the message $\tilde{m}$). Fortunately, when verifying n signatures at once, those pairing operations can be aggregated and performed with the constant 3 parings, regardless of the number n. Of course, n verifications of $\{\pi\}$ should be carried out respectively, but such verifications are all done in $\mathbb{G}_1$. Consequently, all signing and verification operations can be done in $\mathbb{G}_1$, except for the three pairings.

During the interactive join protocol in our $\mathcal{DGS}$ scheme, the primary task of a prospective user is to generate an SPK proof π_0 and a digital signature σ_{DS}. Importantly, our security analysis shows that there is no need to use any knowledge extractor with respect to that proof. More precisely, π_0 is the proof of equality of discrete logarithms (α, s_0, s_1) with the statement that (1) α is the exponent of $f = g^\alpha$ and $w = u^\alpha$ and (2) α is also the exponent of the encrypted message $\hat{g}^\alpha$ of the double ElGamal encryption scheme $\{\hat{g}^{s_b}, \hat{g}^\alpha \hat{Z}_b^{s_b}\}_{b\in\{0,1\}}$ with random values s_0 and s_1. The fact that $f = g^\alpha$ and $w = u^\alpha$ is necessary for generating the randomizable PS signature $v = u^x w^y \in \mathbb{G}_1$, using the issuer's key (x, y). In the original PS problem [42], v is obtained in response to the oracle input (u, α), and therefore it is required to find out α from π_0 using a

knowledge extractor. However, in the modified GPS problem (defined in Section 3), v is obtained in response to the oracle input (u,w), and thus w as is in the statement can be used without knowledge extractor. Here, u is determined by $u = H(f)$ so that u is eventually assigned to the user according to α (chosen by the user). Because of the *statistical* soundness property of π_0, the issuer is convinced that $v = u^x w^y$ becomes $v = u^{x+\alpha y}$ for some secret α known to the joining user. Such a difference in the oracle input determines whether or not to exclude a knowledge extractor in security analysis. Meanwhile, the other statement is necessary for the opener to obtain $\hat{g}^\alpha$ (not α) as an opening trapdoor via decryption, while ensuring the well-formness of the double ElGamal encryption. Indeed, π_0 plus the double ElGamal encryption can be viewed as a variant of the Naor-Yung double encryption technique [39] under the opener's public key $(\hat{Z}_0, \hat{Z}_1)$. The reason why $\hat{g}^\alpha \in \mathbb{G}_2$ is an opening trapdoor is that, given a group signature (u,v,w) and π such that $w = u^\alpha$ for the same α, the opener checks if $e(u, \hat{g}^\alpha) \stackrel{?}{=} e(w, \hat{g})$ for the public parameter $\hat{g}$. If so, the owner of $\hat{g}^\alpha$ is identified as the signer. If such an opening process works, $\hat{g}^\alpha$ (and thus implicitly α) must be uniquely registered, one per each user. To do this, the issuer needs to additionally check if $f = g^\alpha$ is fresh across all join protocol executions so far, thereby forcing all users to join with different α.

5.2 Construction

Let $pp = (\mathcal{G}, H)$ denote a public parameter, where $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \hat{g})$ is a type-3 bilinear group and $H : \{0,1\}^* \to \mathbb{G}_1$ is a cryptographic hash function. Let $\mathsf{DS}=\{\mathsf{DSKg}, \mathsf{DSSig}, \mathsf{DSVf}\}$ be a digital signature scheme. Our $\mathcal{DGS}$ scheme is constructed as follows:

GKg. The group-key generation does the followings.

1. It chooses random $x, y \in \mathbb{Z}_p$ and computes $\hat{X} = \hat{g}^x$, $\hat{Y} = \hat{g}^y$.
2. It chooses random $z_0, z_1 \in \mathbb{Z}_p$ and computes $\hat{Z}_0 = \hat{g}^{z_0}$ and $\hat{Z}_1 = \hat{g}^{z_1}$.
3. The group public key is $gpk = (\hat{X}, \hat{Y}, \hat{Z}_0, \hat{Z}_1)$, and the issuer's secret key is $ik = (x,y)$, and the opener's secret key is $ok = (z_0, z_1)$.

UKg. The user i obtains a user key pair (upk_i, usk_i) by running DSKg.

GJoin. As shown in Figure 3, the user i and the issuer executes the group-joining protocol, using the algorithms Join and Iss, respectively.

1. Join: The user chooses $\alpha, s_0, s_1 \in \mathbb{Z}_p$ randomly, and computes $f = g^\alpha$, $u = H(f)$, $w = u^\alpha$, $\hat{S}_0 = \hat{g}^{s_0}, \hat{S}_1 = \hat{g}^{s_1}$, $\hat{f}'_0 = \hat{g}^\alpha \hat{Z}_0^{s_0}$, $\hat{f}'_1 = \hat{g}^\alpha \hat{Z}_1^{s_1}$.

   (1) The proof π_0 is generated as

   $$\pi_0 = SPK_0\{(\alpha, s_0, s_1) : f = g^\alpha \wedge w = u^\alpha \wedge \hat{S}_0 = \hat{g}^{s_0} \wedge \hat{S}_1 = \hat{g}^{s_1} \wedge \hat{f}'_0 = \hat{g}^\alpha \hat{Z}_0^{s_0} \wedge \hat{f}'_1 = \hat{g}^\alpha \hat{Z}_1^{s_1}\}.$$

   (2) The user obtains σ_{DS} by running $\mathsf{DSSig}(usk_i, \tau)$, where $\tau = e(f, \hat{g})$.

   (3) The tuple $(f, w, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_1, \pi_0, \sigma_{DS})$ is sent to the issuer.
2. Iss: Given the above tuple, the issuer computes $u = H(f)$ and $\tau = e(f, \hat{g})$. It then checks the followings:

   (1) f was not appeared in a previous or current joining session.

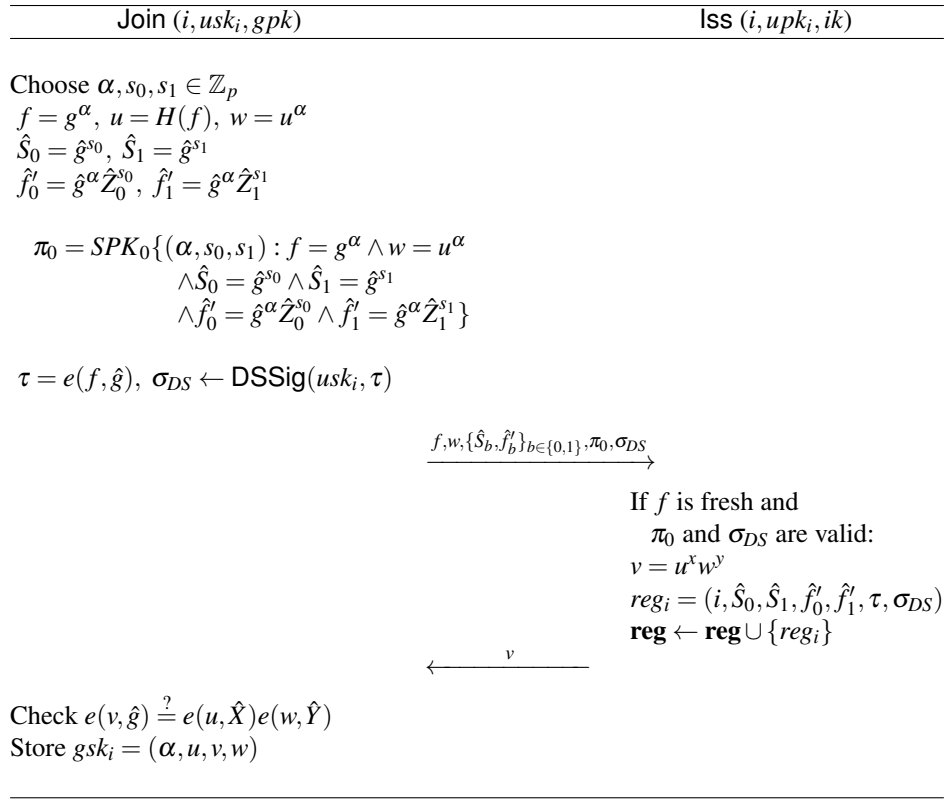

Figure 3: Overview of the GJoin protocol

(2) π_0 is valid on the given tuple.

(3) σ_{DS} is valid on τ under upk_i.

(4) If all conditions are satisfied, then it computes $v = u^x w^y$. It adds $reg_i = (i, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_1, \tau, \sigma_{DS})$ to the list **reg**.

(5) v is sent to the user.

3. Join: If the conditions $e(v, \hat{g}) = e(u, \hat{X}) \cdot e(w, \hat{Y})$ and $u \neq 1$ hold, the user stores the following group signing key
$$gsk_i = (\alpha,\ u,\ v = u^{x+y\alpha},\ w = u^\alpha) \in \mathbb{Z}_p \times \mathbb{G}_1^3.$$

GSig. Given a message m to be signed, the user invokes this algorithm as follows:

1. It chooses $r \in \mathbb{Z}_p$ randomly and computes $u' = u^r$, $v' = v^r$, and $w' = w^r$.
2. π_1 as a proof of knowledge of the user's secret α is generated with
$$\pi_1 = SPK_1\{(\alpha) : w' = u'^\alpha\}(m).$$
3. The group signature is $\sigma = (u',\ v',\ w',\ \pi_1) \in \mathbb{G}_1^3 \times \mathbb{Z}_p^2$.

GVf. A verifier checks if a signature $\sigma = (u', v', w', \pi_1)$ is valid on a message m as follows:

1. It checks if π_1 is valid with respect to (u', w') and m.
2. If so, it checks if $e(v', \hat{g}) = e(u', \hat{X}) \cdot e(w', \hat{Y})$.
3. If both conditions hold, it outputs 1, and otherwise 0.

GOpen. Given a valid $\sigma = (u', v', w', \pi_1)$ on a message m, the opener does as follows:

1. For each $reg_i = (i, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_0, \tau, \sigma_{DS}) \in \mathbf{reg}$,
   (1) it derives $\hat{f} = \hat{f}'_b \cdot (\hat{S}_b^{z_b})^{-1}$ for a randomly chosen bit $b \in \{0,1\}$.
   (2) it checks if $e(u', \hat{f}) = e(w', \hat{g})$ and $\tau = e(g, \hat{f})$.
2. If Step 1 fails for all reg_i, it outputs $\perp$.
3. Otherwise, i.e., if $\hat{f}$ is found for a certain reg_i, π_2 as a proof of knowledge of $\hat{f}$ is generated by
$$\pi_2 = SPK_2\{(\hat{f}) : e(w', \hat{g}) = e(u', \hat{f}) \wedge \tau = e(g, \hat{f})\}.$$
4. It outputs the identity i along with $\Pi = (\tau, \sigma_{DS}, \pi_2)$.

GJudge. Given $(m, \sigma, gpk, i, upk_i, \Pi)$ where $\sigma = (u', v', w', \pi_1)$ is a valid signature and $\Pi = (\tau, \sigma_{DS}, \pi_2)$, this algorithm does as follows:

1. It checks if π_2 is valid.
2. It also checks if $\mathsf{DSVf}(upk_i, \tau, \sigma_{DS}) = 1$.
3. It outputs 1 if the above conditions holds; otherwise 0.

It is easily checked that our $\mathcal{DGS}$ is correct because of the completeness of the SPKs and the correctness of DS. We omit the details.

5.3 Batch Verification

We note that n group signatures from different signers can be verified efficiently via pairing-batch computations. The batch verification algorithm BGVf is described as follows:

BGVf. It takes n group signatures $\{\sigma_i\}_{i=1}^n$. Let ℓ be a small prime and let $\sigma_i = (u'_i, v'_i, w'_i, \pi_{i,1})$ be the i-th group signature on a message m_i. A verifier does as follows:

1. For each $i = 0, \ldots, n$, it checks if $\pi_{i,1}$ is valid on (u'_i, w'_i) and m_i.
2. If so, it randomly chooses $e_1, ..., e_n \in \{0,1\}^\ell$ and computes $\{\tilde{u}_i = u_i'^{e_i}, \tilde{v}_i = v_i'^{e_i}, \tilde{w}_i = w_i'^{e_i}\}_{i=1}^n$.
3. It checks if $e(\prod_{i=1}^n \tilde{v}_i, \hat{g}) = e(\prod_{i=1}^n \tilde{u}_i, \hat{X}) \cdot e(\prod_{i=1}^n \tilde{w}_i, \hat{Y})$.
4. If all checks succeed, it outputs 1; otherwise 0.

We note that this batch verification reduces the most expensive operation, $3n$ pairings, to the constant 3 pairings. In the first step, however, n proofs $\{\pi_{i,1}\}_{i=1}^n$ can not be batched and must be verified respectively. Fortunately, all of those operations necessary for verifying the SPKs $\{\pi_{i,1}\}_{i=1}^n$ are performed over $\mathbb{G}_1$ with relatively smaller-sized group elements, compared to $\mathbb{G}_2$ and $\mathbb{G}_T$. In addition, our pairing-batch computation is performed with re-randomization of signatures by employing *small exponents test* [4]. In Section 7, we will give efficiency comparison between previous group signature schemes and ours.

6 Security Proofs

We now prove that our $\mathcal{DGS}$ in Section 5 satisfies the security properties of anonymity, non-frameability, and traceability. Throughout our security analysis, let $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \hat{g})$ be a type-3 bilinear group as a public parameter, and let $\mathcal{L}_h, \mathcal{L}_c, \mathcal{L}_{sk}, \mathcal{L}_\sigma$ and $\mathcal{L}_{ch}$ (described in Section 4.2) be initially empty.

6.1 Anonymity

A challenger $\mathcal{C}$ can assume that i_0^* and i_1^* are queried to the SndToU oracle. Let q_j be the number of the SndToU oracle queries. $\mathcal{C}$ picks a random k among $\{1, \ldots, q_j\}$, hoping that the k-th query is related to i_0^*. Let i^* be the identity i_0^* of the k-th query and $reg^* = (i^*, \hat{S}_0^*, \hat{S}_1^*, \hat{f}_0'^*, \hat{f}_1'^*, \tau^*, \sigma_{DS}^*)$ be the entry registered during the execution of SndToU for i^*.

As shown in Figure 1, to prove anonymity with respect to a challenge signature $\sigma^* = (u^*, v^*, w^*, \pi_1^*)$, we use a hybrid argument by chaining the game $\mathbf{G}_0$ into the game $\mathbf{G}_R$. $\mathbf{G}_0$ is the original anonymity game of $\mathbf{Exp}^{Anon-0}_{\mathcal{DGS},\mathcal{A}}(\lambda)$ with respect to the target identity i^*, and $\mathbf{G}_R$ is the game where α^*, as a secret value that identifies i^*, is not leaked from σ^*, reg^*, and all signatures returned in response to the Sig oracle queries. The next hybrid argument is to change the game $\mathbf{G}_R$ to the anonymity game of $\mathbf{Exp}^{Anon-1}_{\mathcal{DGS},\mathcal{A}}(\lambda)$, but we omit it because the next argument is easily done by reversing the first hybrid argument.

Table 1: Hybrid games for proving anonymity of our $\mathcal{DGS}$

	Note	Indistinguishability
$\mathbf{G}_0$	This is the same as $\mathbf{Exp}^{Anon-0}_{\mathcal{DGS},\mathcal{A}}(\lambda)$, where σ^* is generated by the i_0^*'s group signing key.	
$\mathbf{G}_1$	This is the same as $\mathbf{G}_0$, except that $\mathcal{C}$'s proofs of the SPKs are all simulated.	ZK of the SPKs
$\mathbf{G}_2$	This is the same as $\mathbf{G}_1$ except that, in SndToU(i^*), $\mathcal{C}$ sends $\mathcal{A}$ random $\hat{R}_0, \hat{R}_1 \in \mathbb{G}_2$ instead of $\hat{f}_0'^*, \hat{f}_1'^*$ in reg^*, respectively.	$\mathrm{XDH}_{\mathbb{G}_2}$ assumption
$\mathbf{G}_R$	This is the same as $\mathbf{G}_2$ except that w^* in σ^* is chosen at random.	$\mathrm{XDH}_{\mathbb{G}_1}$ assumption, non-fraemability

Lemma 6.1. $\mathbf{G}_0$ *and* $\mathbf{G}_1$ *are indistinguishable under the ZK of the SPKs.*

Proof. By the definition of the ZK in Section 2.3, all proofs can be simulated by the zero-knowledge simulator $\mathcal{S}$, which is statistically indistinguishable from a real one. □

Lemma 6.2. *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then,* $\mathbf{G}_1$ *and* $\mathbf{G}_2$ *are indistinguishable under the* $XDH_{\mathbb{G}_2}$ *assumption.*

Proof. Without loss of generality, we first change $\hat{f}_0'^*$ and sequentially $\hat{f}_1'^*$ at random. To show this, we define an intermediate game (denoted by $\mathbf{G}'$) where a random $\hat{R}_0 \in \mathbb{G}_2$ is replaced with $\hat{f}_0'^*$ in reg^*. We now demonstrate how to change $\mathbf{G}_1$ into $\mathbf{G}'$ as follows.

$\underline{\mathbf{G}_1 \approx \mathbf{G}'}$: $\mathcal{C}$ is given $(\hat{g}, \hat{A} = \hat{g}^a, \hat{B} = \hat{g}^b, T)$, an instance of the $\mathrm{XDH}_{\mathbb{G}_2}$ problem, and uses the zero-knowledge simulator $\mathcal{S}$ of the SPKs. The proof idea is that the first opening key z_0 is set to $b = \log_{\hat{g}} B$ and thus $\hat{Z}_0 = B$. When the user i^* is queried to the SndToU oracle, $\hat{f}_0^* = \hat{g}^\alpha \hat{Z}_0^{s_0}$ is determined by T, by setting $s_0 = a = \log_g \hat{A}$.

Initially, $\mathcal{C}$ chooses $x, y, z_1 \in \mathbb{Z}_p$, and sets $ik = (x, y)$, $ipk = (\hat{X} = \hat{g}^x, \hat{Y} = \hat{g}^y)$, and $opk = (\hat{Z}_0 = \hat{B}, \hat{Z}_1 = \hat{g}^{z_1})$. $\mathcal{C}$ gives $gpk = (ipk, opk)$ and ik to $\mathcal{A}$. The hash oracle Hash-H is managed by returning a random value

for a new input as usual, and thus we omit the details of the hash oracle. $\mathcal{A}$ makes the following queries adaptively.

• SndToU. For a queried identity $i \neq i^*$, $\mathcal{C}$ invokes $\mathsf{UKg}(1^\lambda, i)$ and performs $\mathsf{GJoin}(\mathsf{Join}(i, usk_i, gpk) \leftrightarrow \mathcal{A})$ as in the real scheme, except that π_0 is simulated by $\mathcal{S}$. If $i = i^*$ is queried, $\mathcal{C}$ chooses $\alpha, s_1 \in \mathbb{Z}_p$ randomly and then uses $\mathcal{S}$ to generate π_0 for the following elements

$$(f = g^\alpha, w = H(f)^\alpha,\ \hat{S}_0 = \hat{A},\ \hat{S}_1 = \hat{g}^{s_1},\ \hat{f}'_0 = \hat{g}^\alpha \cdot T,\ \hat{f}'_1 = \hat{g}^\alpha \cdot \hat{Z}_1^{s_1}).$$

The above elements and (π_0, σ_{DS}) is given to $\mathcal{A}$, where σ_{DS} is a digital signature on $\tau = e(f, \hat{g})$. At the end of the protocol, $\mathcal{C}$ receives v from $\mathcal{A}$ and stores $gsk_i = (\alpha, u, v, w)$. The list of honest users is updated as $\mathcal{L}_h \leftarrow \mathcal{L}_h \cup \{i\}$.

• USK. For an identity $i \in \mathcal{L}_h$, $\mathcal{C}$ aborts if $i = i^*$. Otherwise, $\mathcal{C}$ outputs usk_i and gsk_i and updates $\mathcal{L}_{sk} \leftarrow \mathcal{L}_{sk} \cup \{i\}$.

• WReg. For an identity i and an entry ρ, $\mathcal{C}$ sets $reg_i = \rho$.

• Sig. For an identity $i \in \mathcal{L}_h$ and a message m, $\mathcal{C}$ obtains a group signature σ by performing $\mathsf{GSig}(gsk_i, m)$, except for a simulated proof π_1. $\mathcal{C}$ outputs the signature and updates $\mathcal{L}_\sigma \leftarrow \mathcal{L}_\sigma \cup \{(i, m, \sigma)\}$.

• Open. For a message m and a signature σ such that $(*, m, \sigma) \notin \mathcal{L}_{ch}$, $\mathcal{C}$ can open the identity i that produced σ, using the other opening key z_1.

• CH_0. Given identities $i_0^*, i_1^* \in \mathcal{L}_h \backslash \mathcal{L}_{sk}$ and a message m^*, $\mathcal{C}$ aborts if $i^* \neq i_0^*$. Otherwise, $\mathcal{C}$ outputs the challenge signature σ^* generated by $\mathsf{GSig}(gsk_{i_0^*}, m^*)$ and updates $\mathcal{L}_{ch} \leftarrow \mathcal{L}_{ch} \cup \{(i_0^*, m^*, \sigma^*), (i_1^*, m^*, \sigma^*)\}$.

The probability that $\mathcal{C}$ does not abort is at least $1/q_j$. $\mathcal{C}$ simulates $\mathbf{G}_1$ if $T = g^{ab}$; otherwise, it simulates $\mathbf{G}'$. Hence, as long as $\mathcal{C}$ does not abort, $\mathcal{C}$ can solve the $\mathrm{XDH}_{\mathbb{G}_2}$ problem, using $\mathcal{A}$.

$\underline{\mathbf{G}' \approx \mathbf{G}_2}$: Similarly, $\mathcal{C}$ changes $\hat{f}'^*_1$ of reg^* into a random $\hat{R}_1 \in \mathbb{G}_2$ under the $\mathrm{XDH}_{\mathbb{G}_2}$. The simulation for this change is almost similar to the previous one, except that $\mathcal{C}$ chooses the first opening key z_0 by itself and sets the second key to be the unknown $z_1 = b = \log_{\hat{g}} \hat{B}$. Because of the distinction, when dealing with the SndToU oracle for i^*, $\mathcal{C}$ chooses $\alpha, s_0 \in \mathbb{Z}_p$ randomly and then uses $\mathcal{S}$ to generate π_0 for the following elements

$$(f = g^\alpha,\ w = H(f)^\alpha,\ \hat{S}_0 = \hat{g}^{s_0},\ \hat{S}_1 = \hat{A},\ \hat{f}'_0 = \hat{R}_0,\ \hat{f}'_1 = \hat{g}^\alpha \cdot T),$$

where $\hat{R}_0 \in \mathbb{G}_2$ are randomly chosen by $\mathcal{C}$ and the values $\hat{A}, T$ come from the instance of the $\mathrm{XDH}_{\mathbb{G}_2}$ problem.

In response to the Open oracle, $\mathcal{C}$ can open the identity i that produced σ, using the other opening key z_0. Additionally, given (m, σ) that would be generated by i^*, $\mathcal{C}$ can use $\hat{g}^\alpha$ and pairing checks to open the target identity i^*.

Similarly to the previous proof, the probability that $\mathcal{C}$ does not abort is at least $1/q_j$. $\mathcal{C}$ simulates $\mathbf{G}'$ if $T = g^{ab}$; otherwise, it simulates $\mathbf{G}_2$. Hence, as long as $\mathcal{C}$ does not abort, $\mathcal{C}$ can solve the $\mathrm{XDH}_{\mathbb{G}_2}$ problem, using $\mathcal{A}$.

□

Lemma 6.3. *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then* $\mathbf{G}_2$ *and* $\mathbf{G}_R$ *are indistinguishable under the* $XDH_{\mathbb{G}_1}$ *assumption and non-frameability.*

Proof. Let $(g, A = g^a, B = g^b, T)$ be an instance of the $\mathrm{XDH}_{\mathbb{G}_1}$ problem, and let $\mathcal{S}$ be the zero-knowledge simulator given to a challenger $\mathcal{C}$. The proof idea is that the secret α^* of the target user i^* is set to $b = \log_g B$ and the random r^* chosen in CH_0 is set to $a = \log_g A$.

$\mathcal{C}$ sets $ik = (x, y)$ and $ok = (z_1, z_2)$, where $x, y, z_1, z_2 \in \mathbb{Z}_p$ are chosen by itself, and gives $gpk = (\hat{X} = \hat{g}^x, \hat{Y} = \hat{g}^y, \hat{Z}_1 = \hat{g}^{z_1}, \hat{Z}_2 = \hat{g}^{z_2})$ and ik to $\mathcal{A}$. The Hash-H oracle is managed with a hash list $\mathcal{L}_H$ as follows:

• Hash-H. For a queried input n, if n is new, $\mathcal{C}$ computes $\zeta = g^{\delta_n}$ by choosing a random $\delta_n \in \mathbb{Z}_p$. $\mathcal{C}$ returns ζ and updates $\mathcal{L}_H \leftarrow \mathcal{L}_H \cup \{(n, \delta_n, \zeta)\}$. If n was already used, $\mathcal{C}$ returns a previously computed ζ corresponding to n in $\mathcal{L}_H$.

• SndToU. If $i \neq i^*$, $\mathcal{C}$ obtains from UKg a user key pair (usk_i, upk_i). By using the keys, it executes GJoin(Join$(i, usk_i, gpk) \leftrightarrow \mathcal{A}$), except for simulating π_0. If $i = i^*$, $\mathcal{C}$ executes the joining protocol without knowing $\alpha^* = b = \log_g B$. At the end of the protocol, $\mathcal{A}$ is given the following elements

$$(f = B,\ w^* = B^{\delta_n^*},\ \hat{S}_0 = \hat{g}^{s_0},\ \hat{S}_1 = \hat{g}^{s_1},\ \hat{f}_0' = \hat{R}_0,\ \hat{f}_1' = \hat{R}_1,\ \pi_0,\ \sigma_{DS}),$$

where δ_n^* is the value associated with f in $\mathcal{L}_H$, and $s_0, s_1 \in \mathbb{Z}_p$ and $\hat{R}_0, \hat{R}_1 \in \mathbb{G}_2$ are randomly chosen, and π_0 is a simulated proof by using $\mathcal{S}$, and σ_{DS} is a digital signature on $\tau = e(B, \hat{g})$. Eventually, $\mathcal{C}$ stores $gsk_i = (\alpha, u, v, w)$ for all users except i^*, and stores $gsk_{i^*} = (\perp, u^*, v^*, w^*)$ for i^* where $u^* = g^{\delta_n^*}$ and $v^* = u^{*x} w^{*y}$. The list of honest users is updated as $\mathcal{L}_h \leftarrow \mathcal{L}_h \cup \{i\}$.

• USK. For an identity $i \in \mathcal{L}_h$, $\mathcal{C}$ sends usk_i and gsk_i to $\mathcal{A}$ and updates $\mathcal{L}_{sk} \leftarrow \mathcal{L}_{sk} \cup \{i\}$. If $i = i^*$, $\mathcal{C}$ aborts.

• WReg. For an identity i and an entry ρ, $\mathcal{C}$ sets $reg_i = \rho$.

• Sig. If $i \neq i^*$ such that $i \in \mathcal{L}_h$, a group signature σ on a message m can be produced by performing GSig except for a simulated proof π_1. If $i = i^*$, $\mathcal{C}$ uses $gsk_{i^*} = (\perp, u^*, v^*, w^*)$ to compute a signature $\sigma = (u^{*r}, w^{*r}, v^{*r}, \pi_1)$, where $r \in \mathbb{Z}_p$ is chosen randomly and the proof π_1 is generated as a simulated proof of knowledge of $\alpha^* = b$. $\mathcal{C}$ outputs σ to $\mathcal{A}$ and updates $\mathcal{L}_\sigma \leftarrow \mathcal{L}_\sigma \cup \{(i, m, \sigma)\}$.

• Open. Given a message m and a signature σ such that $(*, m, \sigma) \notin \mathcal{L}_{ch}$, $\mathcal{C}$ uses either $\mathcal{L}_\sigma$ or one of the opening keys to open (m, σ). If σ is generated by a non-target user, $\mathcal{C}$ performs the GOpen algorithm and returns the result. However, if σ is generated by i^* and $(i^*, m, \sigma) \in \mathcal{L}_\sigma$, $\mathcal{C}$ uses $\mathcal{L}_\sigma$ to realize that σ was produced in response to the oracle Sig(i^*, m). The problem is the case when σ is generated by i^* and $(i^*, m, \sigma) \notin \mathcal{L}_\sigma$. However, this happens when $\mathcal{A}$ forges a valid signature for i^*, giving rise to breaking the non-frameability. Thus, by proving Theorem 6.5, such a troublesome case can be excluded.

• CH$_0$. Given two identities $i_0^*, i_1^* \in \mathcal{L}_h \backslash \mathcal{L}_{sk}$ and a message m, $\mathcal{C}$ aborts if $i_0^* \neq i^*$. Otherwise, using $gsk_{i^*} = (\perp, u^*, v^*, w^*)$ and $ik = (x, y)$, $\mathcal{C}$ computes a challenge signature as below.

$$\sigma^* = (u'^* = A^{\delta_n^*}, \quad v'^* = (u'^*)^x \cdot (w'^*)^y, \quad w'^* = T^{\delta_n^*}, \quad \pi_1^*),$$

where $r^* = a = \log_g A$ is used to randomize (u^*, v^*, w^*) and π_1^* is generated as a simulated proof of knowledge $\alpha^* = b$. $\mathcal{C}$ outputs σ^* and updates $\mathcal{L}_{ch} \leftarrow \mathcal{L}_{ch} \cup \{(i^*, m^*, \sigma^*), (i_1^*, m^*, \sigma^*)\}$.

Then, the probability that $\mathcal{C}$ does not abort is at least $1/q_j$. $\mathcal{C}$ simulates $\mathbf{G}_2$ if $T = g^{ab}$; otherwise it simulates $\mathbf{G}_R$. Therefore, as long as $\mathcal{C}$ does not abort, $\mathcal{C}$ can solve the XDH$_{\mathbb{G}_1}$ problem by using $\mathcal{A}$.

□

By putting the Lemmas 6.1, 6.2, and 6.3 and considering the second hybrid argument (we omitted) all together, we can get the following result.

Theorem 6.4. *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then our $\mathcal{DGS}$ in Section 5 is anonymous under the ZK of the SPKs, the SXDH assumption, non-frameability of $\mathcal{DGS}$.*

6.2 Non-frameability

Theorem 6.5. *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then our $\mathcal{DGS}$ in Section 5 is non-frameable under the SS of the SPK_2, the EUF-CMA of the underlying digital signature scheme, and the SDL assumption.*

Proof. Let $(g, \hat{g}, D = g^d, \hat{D} = \hat{g}^d)$ be an instance of the SDL problem, and let $\mathcal{S}$ and $\mathcal{E}$ be the zero-knowledge simulator and the knowledge extractor for the SPKs, respectively. Also, let $\mathsf{DS} = \{\mathsf{DSKg}, \mathsf{DSSig}, \mathsf{DSVf}\}$ be a digital signature scheme.

A challenger $\mathcal{C}$ can assume that a target user i^* is queried to the SndToU oracle. Let q_j be the number of the SndToU oracle queries. $\mathcal{C}$ picks a random k among $\{1, \ldots, q_j\}$, hoping that the k-th query is related to i^*. For simplicity, let i^* be the identity of the k-th query.

The proof idea is that $\mathcal{C}$ sets the unknown $d = \log_g D = \log_{\hat{g}} \hat{D}$ as a group signing key of the target user i^*. When $\mathcal{A}$ succeeds in forging a signature with respect to d, $\mathcal{C}$ can solve the SDL problem (using $\mathcal{E}$), or break the EUF-CMA of DS, or break the SS of the SPK_2.

$\mathcal{C}$ generates $ik = (x, y)$ and $ok = (z_0, z_1)$, where $x, y, z_0, z_1 \in Z_p^*$ are randomly chosen by itself. $\mathcal{A}$ is given those ik and ok. The Hash-H oracle is managed with a hash list $\mathcal{L}_H$ as follows:

• Hash-H. For a queried input n, if n is new, $\mathcal{C}$ computes $\zeta = g^{\delta_n}$ by choosing a random $\delta_n \in \mathbb{Z}_p$. $\mathcal{C}$ returns ζ and updates $\mathcal{L}_H \leftarrow \mathcal{L}_H \cup \{(n, \delta_n, \zeta)\}$. If n was already used, $\mathcal{C}$ returns a previously computed ζ corresponding to n in $\mathcal{L}_H$.

• SndToU. For a queried identity i, if $i \neq i^*$, $\mathcal{C}$ generates a user key pair (usk_i, upk_i) by DSKg, and then executes $\mathsf{GJoin}(\mathsf{Join}(i, usk_i, gpk) \leftrightarrow \mathcal{A})$ as in the real scheme. If $i = i^*$, $\mathcal{C}$ chooses $s_0, s_1 \in \mathbb{Z}_p$ and finds the value δ_n^* related to D in $\mathcal{L}_H$. It then computes the following tuple

$$(f = D, \quad w = D^{\delta_n^*}, \quad \hat{S}_0 = \hat{g}^{s_0}, \quad \hat{S}_1 = \hat{g}^{s_1}, \quad \hat{f}_0' = \hat{D} \cdot \hat{Z}_0^{s_0}, \quad \hat{f}_1' = \hat{D} \cdot \hat{Z}_0^{s_0}, \quad \pi_0, \quad \sigma_{DS} = \mathsf{DSSig}(usk_{i^*}, \tau)),$$

where $\tau = e(f, \hat{g}) = e(D, \hat{g})$ and π_0 is a simulated proof of knowledge $d = \log_g D = \log_{\hat{g}} \hat{D}$ by using $\mathcal{S}$. At the end of the protocol, $\mathcal{C}$ receives v from $\mathcal{A}$. $\mathcal{C}$ then stores $gsk_i = (\alpha, u, v, w)$ if $i \neq i^*$; otherwise $(\perp, u, v, w)$. $\mathcal{C}$ updates $\mathcal{L}_h \leftarrow \mathcal{L}_h \cup \{i\}$.

• USK. For an identity $i \in \mathcal{L}_h$, $\mathcal{C}$ sends usk_i and gsk_i to $\mathcal{A}$ and updates $\mathcal{L}_{sk} \leftarrow \mathcal{L}_{sk} \cup \{i\}$. If $i = i^*$, $\mathcal{C}$ aborts.

• Sig. For a queried identity $i \in \mathcal{L}_h$ and a message m, if $i \neq i^*$, $\mathcal{C}$ is able to generate a group signature on any message that $\mathcal{C}$ chooses, because $\mathcal{C}$ knows gsk_i. On the other hand, if $i = i^*$, $\mathcal{C}$ can produce a signature by using $\mathcal{S}$, although $\mathcal{C}$ does not know d. The resulting signature σ is given to $\mathcal{A}$, and $\mathcal{C}$ updates $\mathcal{L}_\sigma \leftarrow \mathcal{L}_\sigma \cup \{(i, m, \sigma)\}$.

At the end, $\mathcal{A}$ outputs a forged signature $\sigma^* = (u^*, v^*, w^*, \pi_1^*)$ on a message m^* and also the opening result $(i^*, \Pi^* = (\tau^*, \sigma_{DS}^*, \pi_2^*))$ together. If i^* is not the k-th user, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ proceeds. Since $(m^*, \sigma^*, i^*, \Pi^*)$ should be verified by the GJudge algorithm, $\mathcal{C}$ is convinced that (1) π_2^* is valid and (2) τ^* is verified by σ_{DS}^* using the public key upk_{i^*} for i^*.

Recall that $\tau = e(f, \hat{g}) = e(D, \hat{g})$ and $\sigma_{DS} = \mathsf{DSSig}(usk_{i^*}, \tau)$ are values initially registered by i^* via the SndToU oracle. Also, the statement of Π^* consists of two equations such that $e(w^*, \hat{g}) = e(u^*, \hat{f})$ and $\tau^* = e(g, \hat{f})$ for some witness $\hat{f}$. We can consider three possible cases of the $\mathcal{A}$'s forgery:

1) $\tau \neq \tau^* \wedge \mathsf{DSVf}(upk^*, \tau^*, \sigma_{DS}^*) = 1$. This case indicates that the unforgeability of DS is broken by the forgery σ_{DS}^* on the message τ^*.

2) $\underline{\tau = \tau^* \wedge e(w^*, \hat{g}) = e(u^*, \hat{f}') \wedge \mathsf{DSVf}(upk^*, \tau^*, \sigma^*_{DS}) = 1}$: This case indicates that a false statement with respect to two distinct witnesses $\hat{f}$ and $\hat{f}'$ is proven to be valid, leading to breaking the SS of the SPK_2.

3) $\underline{\tau = \tau^* \wedge e(w^*, \hat{g}) = e(u^*, \hat{f}) \wedge \mathsf{DSVf}(upk^*, \tau^*, \sigma^*_{DS}) = 1}$: This case indicates that, since $\hat{f} = \hat{D}$, the exponent $d = \log_{\hat{g}} \hat{D}$ as a witness is used to prove the statement that $w^* = (u^*)^d$ of π^*_1. In that case, $\mathcal{C}$ can extract d from π^*_1 using $\mathcal{E}$, which is the solution of the SDL problem.

□

6.3 Traceability

Theorem 6.6. *Let all SPKs be simulation-sound extractable NIZKs and let H be modeled as a random oracle. Then our $\mathcal{DGS}$ in Section 5 is traceable under the modified GPS assumption and the SS of the SPK_0.*

Proof. A challenger $\mathcal{C}$, who does not know the issuer's secret key, will use the oracles of the modified GPS problem to act as the honest issuer. We show that $\mathcal{C}$ can solve the problem by using $\mathcal{A}$, as long as $\mathcal{A}$ cannot break the SS of the SPK_0.

Let $\mathcal{C}$ be given an instance of the modified GPS problem, $(\hat{X}, \hat{Y})$ equipped with oracles $\mathcal{O}_0^{MGPS}$ and $\mathcal{O}_1^{MGPS}$. Throughout the proof, $\mathcal{C}$ maintains additional lists $\mathcal{L}_0^{MGPS}$ and $\mathcal{L}_1^{MGPS}$ containing a pair of input/output values to each oracle. Also, $\mathcal{C}$ is allowed to have access to the zero-knowledge simulator $\mathcal{S}$ and the knowledge extractor $\mathcal{E}$ with respect to the SPK_1.

$\mathcal{C}$ sets $ipk = (\hat{X}, \hat{Y})$, meaning that $ik = (x, y)$ such that $\hat{X} = \hat{g}^x$ and $\hat{Y} = \hat{g}^y$ is unknown to $\mathcal{C}$. $\mathcal{C}$ chooses $z_0, z_1 \in \mathbb{Z}_p$ randomly, and provides $\mathcal{A}$ with $gpk = (\hat{X}, \hat{Y}, \hat{Z}_0 = \hat{g}^{z_0}, \hat{Z}_1 = \hat{g}^{z_1})$ and $ok = (z_0, z_1)$. $\mathcal{C}$ manages the following hash oracle with a list $\mathcal{L}_H$.

• Hash-H. For an input $n \in \{0,1\}^*$, if n exists in $\mathcal{L}_H = \{(n, u)\}$, $\mathcal{C}$ outputs u. If $n \in \mathbb{G}_1$ and $(n, *)$ is not in $\mathcal{L}_H$, $\mathcal{C}$ calls $\mathcal{O}_0^{MGPS}$ to obtain $u \in \mathbb{G}_1$. Then, u is added into $\mathcal{L}_0^{MGPS}$ and (n, u) is added into $\mathcal{L}_H$. $\mathcal{C}$ outputs u. Otherwise, $\mathcal{C}$ chooses a random $u \in \mathbb{G}_1$ adds (n, u) into $\mathcal{L}_H$. $\mathcal{C}$ outputs u.

• AddU. For an identity i, $\mathcal{C}$ generates a user key pair (upk_i, usk_i), and executes $\mathsf{GJoin}(\mathsf{Join}(i, usk_i, gpk) \leftrightarrow \mathsf{Iss}(i, upk_i, ik))$ without knowing $ik = (x, y)$. More precisely, $\mathcal{C}$ chooses $\alpha, s_0, s_1 \in \mathbb{Z}_p$ randomly, and obtains $f = g^\alpha$ and $u = H(f)$ (eventually using $\mathcal{O}_0^{MGPS}$), $w = u^\alpha$, $\hat{f}'_0 = \hat{g}^\alpha \hat{Z}_0^{s_0}$, $\hat{f}'_1 = \hat{g}^\alpha \hat{Z}_1^{s_1}$, $\hat{S}_0 = \hat{g}^{s_0}$, $\hat{S}_1 = \hat{g}^{s_1}$, and $\tau = e(f, \hat{g})$. After that, by calling $\mathcal{O}_1^{MGPS}$ on the input (g, u, f, w), $\mathcal{C}$ obtains $v = u^x w^y$. The tuple (g, u, f, w, v) is added into $\mathcal{L}_1^{MGPS}$.

At the end of the protocol, $\mathcal{C}$ sets $gsk_i = (\alpha, u, v, w)$ and $reg_i = (i, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_1, \tau, \sigma_{DS})$, where σ_{DS} is a digital signature on τ under usk_i. $\mathcal{C}$ updates $\mathbf{reg} \leftarrow \mathbf{reg} \cup \{reg_i\}$ and $\mathcal{L}_h \leftarrow \mathcal{L}_h \cup \{i\}$.

• CrptU. For an identity i, $\mathcal{C}$ sets the i's public key to be upk provided from $\mathcal{A}$. $\mathcal{C}$ marks the status of the user i with 'cont', and updates $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{(i, \mathsf{cont})\}$.

• SndToI. For an identity i such that $(i, \mathsf{cont}) \in \mathcal{L}_c$, $\mathcal{C}$ and $\mathcal{A}$ together execute $\mathsf{GJoin}(\mathcal{A} \leftrightarrow \mathsf{Iss}(i, upk_i, ik))$. In the protocol execution, $\mathcal{C}$ first receives $(f, w, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_1, \pi_0, \sigma_{DS})$. $\mathcal{C}$ checks if (1) f is fresh and (2) π_0 and σ_{DS} are valid. In this process, when $\mathcal{A}$ calls H oracle, $\mathcal{C}$ answers the oracle queries, using $\mathcal{O}_0^{MGPS}$. If all conditions are satisfied, $\mathcal{C}$ calls $\mathcal{O}_1^{MGPS}$ on the input (g, u, f, w) to obtain $v = u^x w^y$. Note that this input should be well-formed to $\mathcal{O}_1^{MGPS}$; otherwise, π_0 breaks the SS of the SPK_0. $\mathcal{C}$ computes $reg_i = (i, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_1, \tau, \sigma_{DS})$, where $\tau = e(f, \hat{g})$. During this process, the input/output values that appeared in the calls to $\mathcal{O}_0^{MGPS}$ and $\mathcal{O}_1^{MGPS}$ are stored into $\mathcal{L}_0^{MGPS}$, $\mathcal{L}_1^{MGPS}$, and $\mathcal{L}_H$, appropriately. $\mathcal{C}$ ends the protocol by sending v to $\mathcal{A}$.

Now, $\mathcal{A}$ obtains $gsk_i = (\alpha, u, v, w)$, where α is unknown to $\mathcal{C}$. The point is that α is *uniquely* used across all join protocol executions, because of the freshness check about $f = g^{\alpha}$, and simultaneously because of the SS of the SPK_0. This means that $\hat{f} = \hat{g}^{\alpha}$ embedded into $(\hat{f}'_0, \hat{f}'_1)$ can also be uniquely used for the GOpen algorithm. Finally, the following lists are updated as $\mathbf{reg} \leftarrow \mathbf{reg} \cup \{reg_i\}$, where reg_i is the registration entry for the joined user i as before, and $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{(i, \mathsf{accept})\}$.

Contrary to the previous proofs [10, 42], $\mathcal{C}$ does not need to use the knowledge extractor $\mathcal{E}$ with respect to π_0, thereby allowing to make polynomially-many SndToI oracle queries.

- USK. For an identity $i \in \mathcal{L}_h$, $\mathcal{C}$ outputs usk_i and gsk_i.
- RReg. For an identity i, $\mathcal{C}$ outputs reg_i.

$\mathcal{A}$ outputs a forgery $\sigma^* = (u^*, v^*, w^*, \pi_1^*)$ on a message m^*. By the definition of the traceability game, if $\mathcal{A}$ succeeds, then $\mathsf{GVf}(gpk, m^*, \sigma^*) = 1$ holds and either of the following two cases should hold.

1) $\underline{\mathsf{GOpen}(ok, m^*, \sigma^*, \mathbf{reg}) = \perp}$: This case indicates that for each $\hat{f}$ derived from $reg_i = (i, \hat{S}_0, \hat{S}_1, \hat{f}'_0, \hat{f}'_1, \tau, \sigma_{DS}) \in \mathbf{reg}$, the two pairing checks in the GOpen algorithm[3] are not satisfied, i.e., $e(u^*, \hat{f}) \neq e(w^*, \hat{g})$ or $\tau \neq (g, \hat{f})$. The second inequality would contradict the SS of the SPK_0. Thus, the failure of the GOpen algorithm leads to the first inequality for all $\hat{f}$, which means that $w^* \neq (u^*)^{\alpha}$ for each $\alpha = \log_{\hat{g}} \hat{f}$. Eventually, we see that $\alpha^* = \log_{u^*} w^*$ is new. Based on this fact, $\mathcal{C}$ uses $\mathcal{E}$ (with respect to the SPK_1) to extract α^*, and outputs (u^*, v^*, α^*) as a solution of the modified GPS problem.

2) $\underline{\mathsf{GOpen}(ok, m^*, \sigma^*, \mathbf{reg}) = (i^*, \Pi^*) \wedge \mathsf{GJudge}(m^*, \sigma^*, gpk, i^*, upk^*, \Pi^*) = 0}$: Let Π^* be parsed as $(\tau^*, \sigma_{DS}^*, \pi_2^*)$. Since each user has a *unique* exponent α via the SndToI oracle and $u^* \neq 1$ is assured by the GVf algorithm, the probability that multiple identities are traced by the GOpen algorithm is 0. We assume that there is only one entry $reg^* \in \mathbf{reg}$ which opens the user i^*, given (m^*, σ^*). As the open algorithm runs honestly, $\hat{f}^*$ derived from reg^* has to satisfy $e(u^*, \hat{f}^*) = e(w^*, \hat{g})$ and $\tau^* = (g, \hat{f}^*)$. This implies that the statement of the SPK_2 is true, and thus π_2^* is correctly verified because of the completeness of the SPK_2. In addition, σ_{DS}^* should be verified on a message τ^* under upk^*, because $\mathcal{C}$ was already convinced of its validity when handling the SndToI oracle queries. Consequently, the judge algorithm should output 1.

□

7 Comparison with Previous Schemes

In the following comparison, we use SEP-GS to denote $\mathcal{GS}$ schemes based on the *sign-encrypt-prove* paradigm, and SRP-GS to denote $\mathcal{GS}$ schemes on the *sign-randomize-prove* paradigm. For comparison, we consider the following pairing-based $\mathcal{GS}$ schemes secure in the random oracle model; the $\mathcal{GS}$ scheme by Boneh, Boyen, and Shacham (BBS04 [11]), a variant of this BBS04 for suggesting efficient bath verification (BBS04⋆ [27]), two of the most recent SEP-GS schemes with high efficiency (DP06 [24] and LMP+16 [37]), and the existing SRP-GS schemes (BCN+10 [10], PS16 [42], and DS18 [25]).

Efficiency. The efficiency relies on which pairing-friendly curve is chosen. In general, at the same security level, it is known that type-3 pairings offer efficiency benefits. For example, over the 256-bit BN curve [3] standardized in ISO/IEC 15946-5 [35] (aiming at the 100-bit security level [2]), each group has the following bit-length: $|\mathbb{Z}_p| = 256$, $|\mathbb{G}_1| = 256$, $|\mathbb{G}_2| = 512$, and $|\mathbb{G}_T| = 3072$. Many of recent cryptographic

[3] Recall that the GOpen algorithm is partially corrupt, so that the opening executions must follow the prescribed program.

Table 2: Efficiency comparison between previous $\mathcal{GS}$ schemes and ours

Scheme	Signature size (bytes)	Sign (ms)	Verify (ms)
SEP-GS			
BBS04 [11]	$3\mathbb{G}_1, 6\mathbb{Z}_p$ (288)	$3E_T, 9E_1$ (6.0)	$1P, 3E_T, 2E_2, 8E_1$ (9.3)
BBS04* [27]	$1\mathbb{G}_T, 3\mathbb{G}_1, 6\mathbb{Z}_p$ (672)	$3E_T, 9E_1$ (6.0)	$1P, 3E_T, 2E_2, 8E_1$ (9.3)
DP06 [24]	$4\mathbb{G}_1, 5\mathbb{Z}_p$ (288)	$4E_T, 7E_1$ (6.5)	$2P, 3E_T, 9E_1$ (10.8)
LMP+16 [37]	$7\mathbb{G}_1, 3\mathbb{Z}_p$ (320)	$4P, 2E_T, 13E_1$ (15.7)	$4P, 4E_2, 11E_1$ (15.3)
SRP-GS			
BCN+10 [10]	$3\mathbb{G}_1, 2\mathbb{Z}_p$ (160)	$1E_T, 3E_1$ (2.0)	$4P, 3E_1$ (10.5)
PS16 [42]	$2\mathbb{G}_1, 2\mathbb{Z}_p$ (128)	$1E_T, 2E_1$ (1.7)	$3P, 3E_1$ (8.1)
DS18 [25]	$4\mathbb{G}_2, 3\mathbb{G}_1, 3\mathbb{Z}_p$ (448)	$5E_2, 6E_1$ (4.8)	$5P, 2E_2, 4E_1$ (14.4)
This work	$3\mathbb{G}_1, 2\mathbb{Z}_p$ (160)	$4E_1$ (1.2)	$3P, 2E_1$ (7.8)

∘ For $i \in \{1, 2, T\}$, E_i indicates an exponentiation in $\mathbb{G}_i$, respectively. ∘ P is a pairing operation. ∘ The numbers in parentheses are when using the BN-256 curve from [14].

Table 3: Batch verification comparison between previous $\mathcal{GS}$ schemes and ours

Scheme	Batch Verify	$n = 20$ (Ratio)	$n = 100$ (Ratio)
SEP-GS			
BBS04 [11]	$nP, 3nE_T, 2nE_2, 8nE_1$	186.0ms (9.3)	930.0ms (13.7)
BBS04* [27]	$2P, nE_T, 13nE_1$	104.8ms (5.2)	504.8ms (7.4)
DP06 [24]	$2nP, 3nE_T, 9nE_1$	216.0ms (10.7)	1080.0ms (15.9)
LMP+16 [37]	$4nP, 4nE_2, 11nE_1$	306.0ms (15.2)	1530.0ms (22.5)
SRP-GS			
BCN+10 [10]	$(2n+2)P, 3nE_1$	118.8ms (5.9)	574.8ms (8.4)
PS16 [42]	$3nP, 3nE_1$	162.2ms (8.1)	810.0ms (11.9)
DS18 [25]	$(n+4)P, 2nE_2, 4nE_1$	105.6ms (5.3)	489.6ms (7.2)
This work	$3P, (2n+3^{\S})E_1$	20.1ms (1)	68.1ms (1)

∘ For $i \in \{1, 2, T\}$, E_i indicates an exponentiation in $\mathbb{G}_i$, respectively. ∘ P is a pairing operation. ∘ n is the number of signatures to be batch-verified. ∘ The timing of each verification is estimated over the BN-256 curve from [14]. ∘ § indicates the cost for small exponents test.

protocols with resource-constrained devices have been implemented over such Type-3 pairings to attain better efficiency. To provide efficiency comparison, we adopt the result from [14] where an exponentiation $\{E_1, E_2, E_T\}$ in $\{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T\}$, and a pairing P over the BN-256 curve take about 0.3ms, 0.6ms, 1.1ms, 2.4ms on a 2.9GHz Intel Core i7-3520M CPU, respectively.

Table 2 shows that the SRP-GS schemes generally provide better efficiency than the SEP-GS schemes. Especially, PS16 and ours seem to be outstanding. Compared with PS16, ours contains one more element in $\mathbb{G}_1$ in the signature size. However, this would be a slight increase in the setting of type-3 pairings. Instead, our scheme is slightly more efficient than PS16 in terms of the signing and verification costs. It is because, in ours, the signature generation takes place only in $\mathbb{G}_1$, and the verification does not require any computation in neither $\mathbb{G}_2$ nor $\mathbb{G}_T$. Indeed, when taking the BN-256 curve [14], the signing time of ours is estimated to 1.2 ms and the verifying time of ours to 7.8 ms, which are currently the fastest to the best of our knowledge.

Furthermore, we emphasize that our scheme is more favourable for batch verification than the other compared $\mathcal{GS}$ schemes. As shown in Table 3, BBS04* and ours have the feature that the number of pairings (which would be the dominant task) is constant, regardless of n signatures. Nevertheless, BBS04* suffers

from the fact that it still requires another expensive operation E_T, dependent on n. The other $\mathcal{GS}$ schemes all require a number of parings that increase linearly with the number n. On the other hand, our batch verification requires only constant 3 pairings and about $2n+3$ exponentiations in $\mathbb{G}_1$ for n signatures. This results in the remarkable efficiency improvement for batch verification. In fact, when taking the BN-256 curve [14], ours is 8.1 times faster when $n = 20$, and 11.9 times faster when $n = 100$, compared with PS16.

Table 4: Security comparison between previous $\mathcal{GS}$ schemes and ours

Scheme	Security Model	Anony.	Rewinding in GJoin♣	Assumption♣	Opening
SEP-GS					
BBS04 [11]	BMW	CPA	No GJoin	q-Type	Sound
BBS04* [27]	BMW	CPA	No GJoin	q-Type	Sound
DP06 [24]	BSZ	CCA2	No	q-Type	Sound
LMP+16 [37]	KY	CCA2	Yes	SXDH	Sound
SRP-GS					
BCN+10 [10]	BMW*	CCA-	Yes	Interactive	Sound
PS16 [42]	BMW*	CCA-	Yes	Interactive	Sound
DS18 [25]	BSZ	CCA2	No	GGM	Weak
This work	BSZ	CCA-	No	Interactive	Sound

∘ BMW = (static group, issuer and opener not separated). ∘ BMW* = (dynamic group, issuer and opener not separated).
∘ BSZ, KY = (dynamic group, issuer and opener separated). ∘ ♣ relates to the security proof of the traceability.

Security. Table 4 presents the security comparison between the previous $\mathcal{GS}$ schemes and ours. Among the $\mathcal{GS}$ schemes supporting dynamic groups, LMP+16, BCN+10, and PS16 require knowledge extractions (using rewinding) to simulate GJoin protocol executions in their traceability proofs. In contrary, DP06 uses a knowledge extractor without any rewinding, and DS18 and ours do not require it so that they support efficient and concurrently-secure GJoin protocols. Regarding security assumptions for the traceability, BBS04, BBS04*, and DS06 are based on q-type assumptions where q is the number of the SndToI oracle queries. LMP+16 relies on the SXDH assumption, and DS18 relies on a structure preserving signature on equivalence classes [33] that are secure in the generic group model. The other schemes (including ours) are based on interactive assumptions such as the LRSW [38] in BCN+10, and the PS [42] in PS16, and the modified PS assumption in ours. In terms of anonymity, DS06, LMP+16 and DS18 provide the full (CCA2) anonymity by allowing both opening queries and group signing key queries even for target users, and BBS04 and BBS04* provide the weaker CPA anonymity by disallowing any opening query. BCN+10, PS16, and ours provide the slightly weaker type of the selfless (CCA-) anonymity only by disallowing signing key queries for target users. However, compared to other anonymity notions [1, 18], the selfless anonymity is generally considered as an acceptable level of anonymity for practical use. Regarding the soundness of the GOpen algorithm, DS18 provides the weak opening soundness [44], which realistically has the weakness that a group of malicious users can share one and the same opening trapdoor without being detected by the issuer, and thus the GOpen algorithm cannot identify the user who created a signature in question. The other $\mathcal{GS}$ schemes except for DS18, however, provide the opening soundness[4] by which the GOpen algorithm can always find the user as the owner of a valid signature.

[4]This is slightly different from the ordinary opening soundness in [44].

8 Proofs of the Assumptions

We recall the Schwarz-Zippel lemma and present the proof of the modified GPS assumptions in the generic group model [45]. Based on the proof result, we can easily prove that the GPS assumption also holds.

Lemma 8.1 (Schwarz-Zippel Lemma)**.** *Let $P \in \mathbb{F}[x_1,...,x_n]$ be a non-zero polynomial of total degree $d \geq 0$ over a field $\mathbb{F}$. If the values $r_1,...,r_n$ are independently chosen at random from a finite subset $S \subset \mathbb{F}$, then $Pr[P(r_1,...,r_n) = 0] \leq \frac{d}{|S|}$.*

Theorem 8.2. *The modified GPS assumption described in Section 3 holds in the generic group model.*

Proof. A simulator $\mathcal{B}$ maintains lists, for $i \in \{1,2,T\}$, $\mathcal{L}_{\mathbb{G}_i} = \{(F_{i,j}, \gamma_{i,j}) : j = 0,...,\eta_i - 1\}$, where $\gamma_{i,j} \in \{0,1\}^*$ is an encoded value of $F_{i,j}$, and η_i is the maximum number of encoded values that an adversary $\mathcal{A}$ can obtain.

$\mathcal{B}$ initially sets the values as $F_{1,0} = 1$, $F_{2,0} = 1$, $F_{2,1} = x$, $F_{2,2} = y$, and $F_{T,0} = 1$, and gives the assigned values $\gamma_{1,0}$, $\gamma_{2,0}$, $\gamma_{2,1}$, $\gamma_{2,2}$, and $\gamma_{T,0}$ to $\mathcal{A}$. It then sets the counter values as $\theta_1 = \theta_T = 1$, and $\theta_2 = 3$. If θ_i eventually reaches η_i during the following simulation, the corresponding oracle does not work.

• Group Operation. $\mathcal{A}$ asks either multiplication or division while giving $\gamma_{i,j}$ and $\gamma_{i,k}$ where $i \in \{1,2,T\}$ and $j,k < \theta_i$. $\mathcal{B}$ then computes $F_{i,\theta_i} = F_{i,j} \pm F_{i,k}$ for the queried operation. If F_{i,θ_i} previously existed as $F_{i,l}$ for some $l < \theta_i$, $\mathcal{B}$ sets $\gamma_{i,\theta_i} = \gamma_{i,l}$; otherwise, chooses γ_{i,θ_i} randomly. $\mathcal{B}$ outputs γ_{i,θ_i} to $\mathcal{A}$, and then updates $\mathcal{L}_{\mathbb{G}_i} \leftarrow \mathcal{L}_{\mathbb{G}_i} \cup \{(F_{i,\theta_i}, \gamma_{i,\theta_i})\}$ and $\theta_i \leftarrow \theta_i + 1$.

• Pairing Operation. For $\gamma_{1,j}$ and $\gamma_{2,k}$ where $j < \theta_1$ and $k < \theta_2$, $\mathcal{B}$ computes $F_{T,\theta_T} = F_{1,j} \cdot F_{2,k}$. If F_{T,θ_T} previously existed as $F_{T,l}$ for some $l < \theta_T$, $\mathcal{B}$ sets $\gamma_{T,\theta_i} = \gamma_{T,l}$; otherwise, chooses γ_{T,θ_T} randomly. $\mathcal{B}$ outputs γ_{T,θ_T} to $\mathcal{A}$, and then updates $\mathcal{L}_{\mathbb{G}_T} \leftarrow \mathcal{L}_{\mathbb{G}_T} \cup \{(F_{T,\theta_T}, \gamma_{T,\theta_T})\}$ and $\theta_T \leftarrow \theta_T + 1$.

• $\mathcal{O}_0^{MGPS}$. Let n be the maximum number of the $\mathcal{O}_0^{MGPS}$ oracle queries. For the k-th query such as $k \leq n$, $\mathcal{B}$ chooses $r_k \in \mathbb{Z}_p$ randomly and sets $F_{1,\theta_1} = r_k$. If F_{1,θ_1} previously existed as $F_{1,l}$ for some $l < \theta_1$, $\mathcal{B}$ sets $\gamma_{1,\theta_1} = \gamma_{1,l}$; otherwise, $\mathcal{B}$ sets chooses γ_{1,θ_1} randomly and outputs γ_{1,θ_1} to $\mathcal{A}$. $\mathcal{B}$ then updates $\mathcal{L}_{\mathbb{G}_1} \leftarrow \mathcal{L}_{\mathbb{G}_1} \cup \{(F_{1,\theta_1}, \gamma_{1,\theta_1})\}$ and $\theta_1 \leftarrow \theta_1 + 1$.

• $\mathcal{O}_1^{MGPS}$. For $(\gamma_{1,0}, \gamma_{1,i_u}, \gamma_{1,i_f}, \gamma_{1,i_w})$ where $i_u, i_f, i_w < \theta_1$, $\mathcal{B}$ checks the validity by the following conditions; (1) γ_{1,i_u} was not queried to this oracle before, (2) γ_{1,i_u} has appeared in $\mathcal{O}_0^{MGPS}$ as output, and (3) the equality $F_{1,i_w} = F_{1,i_u} F_{1,i_f}$ holds. If all conditions are satisfied, $\mathcal{B}$ computes $F_{1,\theta_1} = xF_{1,i_u} + yF_{1,i_w}$. If F_{1,θ_1} previously existed as $F_{1,l}$ for some $l < \theta_1$, $\mathcal{B}$ sets $\gamma_{1,\theta_1} = \gamma_{1,l}$; otherwise (i.e., if F_{1,θ_1} is new), $\mathcal{B}$ chooses γ_{1,θ_1} randomly. $\mathcal{B}$ outputs γ_{1,θ_1} and updates $\mathcal{L}_{\mathbb{G}_1} \leftarrow \mathcal{L}_{\mathbb{G}_1} \cup \{(F_{1,\theta_1}, \gamma_{1,\theta_1})\}$ and $\theta_1 \leftarrow \theta_1 + 1$. Otherwise, $\mathcal{B}$ returns $\perp$. Additionally, to check the validity of $\mathcal{A}'s$ output later, $\mathcal{B}$ maintains a list $\mathcal{L}^*$ that is updated as $\mathcal{L}^* \leftarrow \mathcal{L}^* \cup \{(F_{1,i_u}, F_{1,i_w})\}$ after $\mathcal{B}$ returns v.

The goal of the above MGPS-related oracles is to ensure that $F_{1,i_f} = \alpha F_{1,0}$ and $F_{1,i_w} = \alpha F_{1,i_u}$ for some polynomial $\alpha \in \mathbb{Z}_p[x,y,r_1,\ldots,r_n]$ of $\mathcal{A}$'s choice. In this case, $\mathcal{A}$ obtains a value from the $\mathcal{O}_1^{MGPS}$ oracle as the encoded value of $F_{1,\theta_1} = F_{1,i_u}(x + y\alpha)$ for some α (known to $\mathcal{A}$).

At the end, $\mathcal{A}$ collects encoded values of elements in $\mathbb{G}_1$ via those oracles; Group Operation, $\mathcal{O}_0^{MGPS}$, and $\mathcal{O}_1^{MGPS}$. It is obvious that elements of the other groups $\mathbb{G}_2$ and $\mathbb{G}_T$ do not help produce a tuple whose elements are in $\mathbb{G}_1$. Thus, $\mathcal{A}$ has to combine previously obtained values in $\mathbb{G}_1$ to produce a valid $(\gamma^*_{1,i_u}, \gamma^*_{1,i_v}, m^*)$, where $i_u, i_v < \eta_1$, satisfying (1) $F^*_{1,i_u} \neq 0$, (2) $F^*_{1,i_v} = F^*_{1,i_u}(x + ym^*)$, and (3) $F_{1,i_w} \neq m^* F_{1,i_u}$ for all $(F_{1,i_u}, F_{1,i_w}) \in \mathcal{L}^*$. We now show that $\mathcal{A}$ cannot symbolically succeed in producing such a tuple.

Let r_i be an output in response to the $\mathcal{O}_0^{MGPS}$ oracle queries for $i = 1,\ldots,n$, and let $P_{i,j} \in \mathbb{Z}_p[x,y,r_1,...,r_n]$ be a polynomial associated with $F_{i,j}$. If the $\mathcal{A}'s$ output $(\gamma^*_{1,i}, \gamma^*_{1,j}, m^*)$ is valid, the polynomials corresponding to both $\gamma^*_{1,i}$ and $\gamma^*_{1,j}$ can be represented by $P^*_{1,i} = a_1 + \sum_{k=1}^{n} \left(b_{k,1} r_k + c_{k,1}(x + y\alpha_k) r_k\right)$ and $P^*_{1,j} =$

$a_2 + \sum_{k=1}^{n} \left(b_{k,2} r_k + c_{k,2}(x + y\alpha_k) r_k\right)$ for some $a_1, a_2, \{\alpha_k, b_{k,1}, b_{k,2}, c_{k,1}, c_{k,2}\}_{k=1}^{n}$ in $\mathbb{Z}_p$, which are known to $\mathcal{A}$. Moreover, the equation $P^*_{1,j} = P^*_{1,i} \cdot (x + ym^*)$ should hold, that is:

$$a_2 + \sum_{k=1}^{n} \left(b_{k,2} r_k + c_{k,2}(x + y\alpha_k) r_k\right) = a_1 x + \sum_{k=1}^{n} \left(b_{k,1} r_k x + c_{k,1}(x^2 + xy\alpha_k) r_k\right) + m^* \left(a_1 y + \sum_{k=1}^{n} \left(b_{k,1} r_k y + c_{k,1}(xy + y^2 \alpha_k) r_k\right)\right).$$

Since the two polynomials are equal, each monomial on both sides is also same. Hence, we have the following: $c_{k,1} = 0$ for all k, as no monomial of degree 3 exists on the left side. $a_1 = 0$ as neither of monomial x nor y exist on the left side. $b_{k,2} = 0$ for all k, as no monomial r_k exists on the right side. $a_2 = 0$ as no constant exists on the right side. $c_{k,2} = b_{k,1}$ for all k, due to the monomial xr_k on each side.

With the above fact, the equation can be simplified as follows:

$$\sum_{k=1}^{n} \alpha_k d_k r_k y = m^* \sum_{k=1}^{n} d_k r_k y, \quad \text{where } c_{k,2} = b_{k,1} = d_k.$$

Since $P^*_{1,i} \neq 0$ and $a_1, c_{k,1}, ..., c_{n,1}$ are all zero, there is at least one non-zero $b_{k,1} = d_k$ for some k, leading to $m^* = \alpha_k$. Consequently, for one of the previous tuple (F_{1,u_k}, F_{1,w_k}) in $\mathcal{L}^*$, m^* meets $F_{1,w_k} = m^* F_{1,u_k}$. This implies the invalidity of the $\mathcal{A}$'s output.

Next, the simulation by $\mathcal{B}$ is complete with overwhelming probability. This means that two different polynomials should not be evaluated to the same value. Let $q_i < \eta_i$ be the number of the queries to the Group Operation for $i \in \{1, 2, T\}$. Regarding $\mathbb{G}_1$, every $P_{1,i}$ has degree at most 2 when $\mathcal{A}$ queries to $\mathcal{O}_1^{MGPS}$. $\mathcal{A}$ obtains at most $q_1 + 2n + 1$ encoded elements, and thus there are at most $(q_1 + 2n + 1)^2/2$ pairs of distinct polynomials. By Lemma 8.1, the probability that two of them evaluate the same value is less than $(q_1 + 2n + 1)^2/p$. Similarly, regarding $\mathbb{G}_2$, $P_{2,i}$ has degree at most 1 and $(q_2 + 3)^2/2$ pairs of distinct polynomials exist. Hence, the probability that the collision of evaluation occurs is less than $(q_2 + 3)^2/2p$. Since $P_{T,i}$ has at most degree 3 and $(q_T + 1)^2/2$ pairs possibly exist, the simulation in $\mathbb{G}_T$ is incorrect with probability less then $3(q_T + 1)^2/2p$. It is obvious that the sum of those probabilities is negligible. □

Theorem 8.3. *The GPS assumption described in Section 3 holds if the modified GPS assumption holds.*

Proof. Let $\mathcal{B}$ play the role of both a simulator of the GPS problem and an adversary of the modified GPS problem. We show that $\mathcal{B}$ uses $\mathcal{A}$, an adversary of the GPS problem, to solve the modified GPS problem.

Whenever $\mathcal{A}$ calls $\mathcal{O}_0^{GPS}$ oracle, $\mathcal{B}$ calls $\mathcal{O}_0^{MGPS}$ oracle and relays the output to $\mathcal{A}$. Whenever $\mathcal{A}$ calls $\mathcal{O}_1^{GPS}$ oracle with input (u, m), $\mathcal{B}$ calls the $\mathcal{O}_1^{MGPS}$ oracle by setting $f = g^m$ and $w = u^m$. Notice that $\mathcal{B}$ knows m that is from the $\mathcal{A}$' input. $\mathcal{B}$ receives v as output, and then relays it to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs (u', v', m) such that $v' = u'^{x+ym}$ for a new m that has not been queried. $\mathcal{B}$ outputs the tuple as a solution of the modified GPS solution. Clearly, it is easily shown that the ability of $\mathcal{A}$ to solve the GPS problem can be converted into that of $\mathcal{B}$ to solve the modified GPS problem. □

9 Conclusion

In constructing a $\mathcal{DGS}$ scheme, concurrent joins and batch verifications are features necessary for realistic applications such as vehicular ad-hoc networks. We have constructed the practical $\mathcal{DGS}$ scheme that provides both features. For the proof of concurrent joins, we have introduced the new assumption, called the

modified GPS assumption, thereby removing knowledge extractors while simulating join protocol executions in the traceability proof. In addition, our $\mathcal{DGS}$ scheme supports efficient pairing-batched verifications at the expense of slightly increasing the size of a signature. In terms of computational cost, our construction is able to achieve the fastest signing and (batch) verification, compared with all of the existing pairing-based $\mathcal{GS}$ schemes.

Acknowledgement

We would like to thank Olivier Sanders for pointing out the flaw of batch verification without small exponents test in the earlier version.

References

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.

[2] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *J. Cryptology*, 32(4):1298–1336, 2019.

[3] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford E. Tavares, editors, *SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.

[4] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.

[5] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.

[6] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[7] David Bernhard, Marc Fischlin, and Bogdan Warinschi. Adaptive proofs of knowledge in the random oracle model. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *Lecture Notes in Computer Science*, pages 629–649. Springer, 2015.

[8] David Bernhard, Ngoc Khanh Nguyen, and Bogdan Warinschi. Adaptive proofs have straightline extractors (in the random oracle model). In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 2017*, volume 10355 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2017.

[9] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[10] Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Get shorty via group signatures without encryption. In Juan A. Garay and Roberto De Prisco, editors, *SCN 2010*, volume 6280 of *Lecture Notes in Computer Science*, pages 381–398. Springer, 2010.

[11] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[12] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick D. McDaniel, editors, *CCS 2004*, pages 168–177. ACM, 2004.

[13] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve A. Schneider, editors, *ACNS 2016*, volume 9696 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2016.

[14] Joppe W. Bos, Craig Costello, and Michael Naehrig. Exponentiating in pairing groups. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 438–455. Springer, 2013.

[15] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick D. McDaniel, editors, *CCS 2004*, pages 132–145. ACM, 2004.

[16] Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In Peng Ning and Ting Yu, editors, *WPES 2007*, pages 21–30. ACM, 2007.

[17] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *SP 2017*, pages 901–920. IEEE Computer Society, 2017.

[18] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016*, volume 9615 of *Lecture Notes in Computer Science*, pages 234–264. Springer, 2016.

[19] Jan Camenisch and Jens Groth. Group signatures: Better efficiency and new theoretical aspects. In Carlo Blundo and Stelvio Cimato, editors, *SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2004.

[20] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[21] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[22] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[23] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 1991.

[24] Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In Phong Q. Nguyen, editor, *VIETCRYPT 2006*, volume 4341 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2006.

[25] David Derler and Daniel Slamanig. Highly-efficient fully-anonymous dynamic group signatures. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *AsiaCCS 2018*, pages 551–565. ACM, 2018.

[26] Keita Emura and Takuya Hayashi. Road-to-vehicle communications with time-dependent anonymity: A lightweight construction and its experimental results. *IEEE Trans. Veh. Technol.*, 67(2):1582–1597, 2018.

[27] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 2009.

[28] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[29] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2005.

[30] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2003.

[31] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[32] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2006.

[33] Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 491–511. Springer, 2014.

[34] IEEE. 802.11p-2010 - IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. Standard, IEEE standards association, 2010.

[35] ISO. ISO/IEC 15946-5:2017 Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 5: Elliptic curve generation. Standard, International Organization for Standardization, 2017.

[36] Aggelos Kiayias and Moti Yung. Group signatures with efficient concurrent join. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 198–214. Springer, 2005.

[37] Benoît Libert, Fabrice Mouhartem, Thomas Peters, and Moti Yung. Practical "signatures with efficient protocols" from simple assumptions. In Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang, editors, *AsiaCCS 2016*, pages 511–522. ACM, 2016.

[38] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.

[39] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *STOC 1990*, pages 427–437. ACM, 1990.

[40] Gregory Neven, Gianmarco Baldini, Jan Camenisch, and Ricardo Neisse. Privacy-preserving attribute-based credentials in cooperative intelligent transport systems. In *IEEE Vehicular Networking Conference 2017*, pages 131–138. IEEE, 2017.

[41] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[42] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2016.

[43] Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *J. Comput. Secur.*, 15(1):39–68, 2007.

[44] Yusuke Sakai, Jacob C. N. Schuldt, Keita Emura, Goichiro Hanaoka, and Kazuo Ohta. On the security of dynamic group signatures: Preventing signature hijacking. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 715–732. Springer, 2012.

[45] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[46] William Whyte, André Weimerskirch, Virendra Kumar, and Thorsten Hehn. A security credential management system for V2V communications. In *IEEE Vehicular Networking Conference 2013*, pages 1–8. IEEE, 2013.