Efficient Dynamic Group Signatures with Forward Security

Amin Mohammadali[1][0000−0001−8495−7421] and Riham AlTawy[2][0000−0002−4734−3700]

University of Victoria, BC , Canada
aminmohammadali@uvic.ca raltawy@uvic.ca

Abstract. In dynamic group signature schemes (GSS), forward security ensures that newly joined members cannot generate valid signatures for past time periods. Additionally, non-frameability prevents even privileged entities, such as the group manager or key issuer, from falsely attributing signatures to honest users. Most GSS either lack non-frameability or face significant efficiency challenges when updating signing keys to ensure forward security. In this paper, we introduce a forward-secure dynamic group signature scheme that guarantees non-frameability. We also present an alternative scheme that, while lacking non-frameability, offers higher efficiency compared to existing schemes with comparable security. For both protocols, we propose efficient revocation mechanisms that allow an authority to revoke users without requiring re-registering existing users. Additionally, we propose a technique that enables the verification process of both protocols to be performed in batches. We prove the security of our schemes, ensuring the standard dynamic GSS security notions; anonymity, traceability and non-frameability (second scheme). Experimental results demonstrate that our schemes are competitive in both computational and communication efficiency when compared to existing literature.

Keywords: Dynamic Group Signature, Forward Security, Non-framability, Revocation.

1 Introduction

Group Signature Schemes (GSS) were first introduced by Chaum and van Heyst in 1991 [13] and has since gained substantial attention from the cryptographic community as well as various stakeholders, specifically, those concerned with user privacy in the digital world. In a GSS, each signer (a.k.a member) has a secret signing key to generate signatures that can be verified using the group's public key. A valid group signature authenticates the signer's membership in the group while ensuring their anonymity. However, in the event of misbehavior, an authorized entity known as Group Manager (GM) can identify the signer. Furthermore, a trusted entity has the power to revoke a user's membership such that the remaining members can continue the group signing process [13,

36, 31, 3]. Group signatures, have found extensive applications across multiple domains such as electronic voting systems [30], blockchains [1, 36], anonymous authentication [15, 25], access control systems [40], and vehicle communication systems [38, 22].

Group signatures are classified as static or dynamic. In static groups, a trusted entity known as key issuer assigns unique signing keys to a static number of users, requiring complete trust in the issuer and limiting the scalability of the scheme [6]. On the other hand, in dynamic groups, users participate in generating their own keys using an interactive *Join* procedure with the issuer which reduces the trust assumptions about the key issuer and extends the flexibility of the scheme [6, 23]. Bellare et al. have defined anonymity, traceability and non-framability and have shown that they are required security notions for dynamic GSS [7]. Moreover, they have proven that these three requirements imply other notions such as unforgeability, unlinkability, exculpability, and collusion resistance. Anonymity ensures that a signature does not reveal the signer's identity within the group and prevents linking multiple signatures to the same signer. Traceability ensures that an adversary cannot generate a signature that either prevents the GM from identifying its signer or causes the GM to mistakenly attribute it to an honest member. Non-framability on the other hand guarantees that no entity, even the group manager and key issuer, can falsely generate a signature and convince a judge that it is generated by an honest user.

Key challenges in dynamic schemes include forward security, preventing new users from signing past messages[6], and revocation, ensuring revoked users cannot sign while preserving the validity of other members' signatures. Group signature revocation processes generally fall into two main categories. The first approach involves maintaining a revocation list that records information about excluded members [28, 11]. The second method focuses on updating the secret information of remaining members to prevent the revoked user from participating in future signing [37, 39]. These distinct strategies offer different trade-offs in terms of efficiency and security in managing group membership. In existing schemes that support revocation [10, 3], revoked users are effectively prevented from participating in the signature generation process. However, when new users join the group, they can still generate valid signatures for events that occurred prior to their enrollment. To add forward security, existing approaches typically require all group members to participate in a new join procedure to obtain updated public and private keys. This process imposes substantial computational and communication overhead, rendering it impractical for large-scale systems. To date, there have been proposed a wide range of schemes that do not guarantee efficient forward security [24, 36, 2, 12, 9, 16, 18, 32, 26, 19, 10]. We aim to design an efficient forward secure group signature scheme that ensures standard security properties while achieving high efficiency across all procedures, including signature generation, verification, tracing, and revocation.

Related works. Boneh et al. introduced the BBS scheme which prevents even the key issuer from generating a signature on behalf of a valid user [10]. Subsequently, Boneh et al. proposed a variant with verifier-Local revocation, reducing

communication overhead but increasing computational costs, as verification time grew with the number of revoked users [11]. They suggested a modification to achieve constant-time verification at the expense of slightly reduced anonymity. Ferrara et al. extended BBS by integrating batch verification, but this significantly increased signature size, making it less suitable for constrained environments [19]. Bichsel et al. explored an alternative to BBS but introduced higher computational costs for both signing and verification [9]. Delerablee et al. developed a GSS scheme with stronger security model (CCA-anonymity instead of CPA) and slightly reduced signature size compared to BBS but increased computational overhead [17]. Similarly, Libert et al. proposed a GSS that ensures anonymity but suffers from high computational costs in both signer and verifier sides [26]. Bichsel et al. introduced the Sign-Randomize-Prove (SRP) paradigm, significantly improving the signature size but imposing high computational costs for identity opening and revocation. Specifically, verification required checking an equation for each revoked user, making it impractical for large groups [9]. Pointcheval and Sanders later extended SRP but lacked a revocation mechanism, further limiting its scalability [32]. Derler et al. proposed a group signature scheme, but it remained inefficient in the opening phase and incurred high computational and communication costs [18]. Clarisse et al. improved SRP-based schemes, achieving competitive efficiency but failed to support revocation and efficient opening process [16]. Kim et al. extended the Pointcheval–Sanders (PS) scheme to support batch verification. However, their approach lacked a revocation mechanism [24]. Libert et al. [27] proposed a forward-secure group signature scheme; however, their definition of forward security deviates from that of Bellare et al. [6]. In Libert et al.'s model, the private key is periodically updated, and even if a key is compromised, it cannot be used to derive keys from earlier time periods. In contrast, Bellare et al.'s definition strengthens this notion by additionally requiring that an adversary, even after compromising the current key, must not be able to forge valid signatures for any *past* time periods. Building on these concepts, Ling et al. [29] proposed a forward-secure group signature scheme based on a lattice-based cryptosystem. However, it suffers from substantial communication overhead.

Our Contribution. This paper presents two practical dynamic group signature schemes (GSSs) that address forward security, non-frameability, and computational and communication efficiency. Our schemes ensure efficiency in signature generation, opening, and revocation. We propose two protocols: the first, denoted as DGSS, prioritizes efficiency but does not maintain non-frameability, while the second, denoted as xDGSS, guarantees non-frameability with a slight increase in communication and computation overhead. Additionally, we introduce a batch verification technique that enhances the efficiency of both schemes. We provide a comprehensive security analysis, proving that our protocols satisfy anonymity, traceability, and non-frameability (for xDGSS). Finally, we implement the cryptographic operations of our schemes and, using experimental measurements, compare their performance and security with other

related works.

2 Preliminaries

In what follows, we briefly introduce the cryptographic primitives that are used in the design of our schemes. We also detail the generic description of dynamic group signature schemes and the associated security requirements.

2.1 Bilinear Pairing

Let a bilinear group be denoted by $\{q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2\}$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q, e is a bilinear map such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$. g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Let φ denote a computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$, with $\varphi(g_2) = g_1$. We assume that the φ is one-way (easy to compute $\mathbb{G}_1 = \varphi(\mathbb{G}_2)$, and hard to compute $\mathbb{G}_2$ from $\mathbb{G}_1$.

2.2 Complexity Assumptions

In what follows, we introduce the complexity assumptions used in the proposed group signature.

Definition 1. *(Computational Diffie-Hellman Problem (CDH)[34]): Given $g \in \mathbb{G}$, $X = g^x$ and $Y = g^y$, for unknowns x and y chosen at random from $\mathbb{Z}_q^*$, find Z such that $Z = g^{xy}$.*

Definition 2. *(Strong Diffie-Hellman (SDH) problem [10]): Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order p. Given generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $(g_1, g_2, g_2^x, g_2^{x^2}, \ldots, g_2^{x^q})$ for a randomly chosen $x \in \mathbb{Z}_p^*$, the Strong Diffie-Hellman (SDH) assumption states that no probabilistic polynomial-time adversary $\mathcal{A}$ can compute a pair $(g_1^{1/(x+c)}, c)$, where $c \in \mathbb{Z}_p^*$, with non-negligible probability.*

2.3 Dynamic Group Signature Schemes

A dynamic GSS consists of a tuple of seven probabilistic polynomial-time (PPT) algorithms: KeyGen, Join, Sign, Verify, Open, Judge, and Update.

In the *KeyGen* algorithm, given a security parameter λ, a key generator produces the issuer's secret key(s) isk, the group public key(s) gpk, and the GM/opener's secret key(s) ok. Secret keys are securely delivered to their respective owners, and gpk is published. A user u_i engages in the interactive *Join* algorithm with the issuer to obtain a secret signing key sk_i, after which the issuer stores the user's registration information reg_i in the registration list $Rlist$. Using sk_i, a member signs a message $M \in \{0,1\}^*$ via the *Sign* algorithm to produce a group signature σ. Anyone can run the *Verify* algorithm with gpk, M, and σ to check validity, returning 1 if valid, and 0 otherwise. The GM executes the

Open algorithm using ok, M, σ, and *Rlist* to identify the signer, returning the identity or $\perp$ if no match is found. The *Judge* algorithm allows anyone to verify the correctness of the GM's output, taking M, σ, gpk, and the revealed identity as input, and returning 1 if valid, or 0 otherwise. Finally, the key issuer updates the public keys to ensure that a new member cannot generate a valid signature using the previous public key, while existing members can continue signing with the updated keys.

GSS security notions. Bellare et al. [7] established three essential requirements for a dynamic group signature scheme: anonymity, traceability, and non-framability. They showed that these three requirements guarantee all the necessary GSS requirements defined in the literature, including unlinkability, unforgeability, collusion resistance [4], strong exculpability [10], and framing resistance [14]. We briefly explain these properties and refer the reader to the original paper [7] for details. Moreover, we formally define the notion of forward security in dynamic group signatures, based on the framework presented in Bellare et al. [6].

Anonymity. This requirement ensures that no adversary can identify the signer or link signatures to the same signer. The anonymity model for dynamic GSSs allows adversaries to collude with a malicious key issuer, while the GM is assumed to be honest [7]. In this model, an adversary $\mathcal{A}$ queries the anonymity challenger with two uncorrupted identities (i_0^*, i_1^*) and a message m^*. The challenger responds with a signature σ_b^* under i_b^* for a randomly chosen bit $b \in \{0, 1\}$. If $\mathcal{A}$ can distinguish which identity was used, it successfully breaks anonymity by returning $b' = b$. Note that the (m^*, σ^*) tuple is restricted from being queried to the open oracle. The adversary can leverage an oracle to obtain secret keys for all users except i_0^* and i_1^*.

Traceability. This requirement ensures that an adversary cannot generate a valid signature that prevents the *Open* algorithm from identifying the actual signer. Additionally, it guarantees that if the *Open* algorithm identifies the signer, the *Judge* algorithm must successfully verify this identity. In this model, the adversary gains access to the GM's key by compromising it, while GM remains an honest entity. The adversary also obtains the secret values of all users except the target user it intends to attack. The adversary $\mathcal{A}$ is allowed to query the sign oracle with messages of its choice and receives a valid signature σ. $\mathcal{A}$ is also allowed to query the open oracle with the signatures and receive valid identifier i. $\mathcal{A}$ succeeds if it outputs a valid σ^* for a message m^* such that m^* has not been queried before and the *Open* algorithm fails to output an identifier of a valid member or it outputs an identifier of an honest member which the *Judge* algorithm fails to verify.

Non-frameability: Non-frameability ensures that an adversary cannot produce a valid signature σ^*, m^* and i^* that convinces the *Judge* algorithm that an honest user created it. The adversary is extremely powerful as it can fully corrupt both the GM/opener and key issuer, obtaining their secret keys. In this model, the adversary $\mathcal{A}$ outputs a message m^*, a signature σ^*, and an identity i^*. $\mathcal{A}$ succeeds if σ^* is a valid signature for m^*, i^* is an honest user, and the *Judge*

algorithm accepts that i^* created σ^*. This must occur without the adversary querying the signing oracle with i^* and m^* or obtaining i^*'s signing key.

Forward Security: Intuitively, a forward-secure group signature scheme ensures that a member cannot generate valid signatures for any previous time period in which they were not a group member [6]. An alternative definition proposed in [35] interprets forward security as the inability of an adversary to efficiently recover past keys from the current one. However, this notion is insufficient, as it does not fully capture the practical security requirements. In a more rigorous model, the adversary, who may be a legitimate group member, is given the current secret key of the targeted signer, as well as the current and previous keys of all other group members. The adversary is considered successful if it can forge a valid signature by the target signer for a document or message corresponding to an earlier time period, i.e., the signature is valid under a group public key that was valid during that previous time period.

3 The Proposed Dynamic Group Signature Schemes

We propose two protocols. The first scheme, DGSS, prioritizes efficiency at the cost of non-framability. The second protocol, xDGSS, ensures non-framability with a slight increase in computational and communication overhead.

3.1 Design Rationale and Technical Contribution

Our scheme builds upon the efficient BBS group signature scheme by Boneh et al. [10], but introduces several key modifications to achieve forward security and non-frameability simultaneously, batch verification, CCA-anonymity, and reduced computational and communication overhead. The original BBS scheme relies on a malleable linear encryption scheme to provide anonymity, which makes it vulnerable to chosen-ciphertext attacks (i.e., it cannot achieve CCA-anonymity). To address this, we replace the linear encryption with an ElGamal encryption augmented by an additional signature element to ensure that tampering with ciphertexts can be detected. This modification allows us to achieve CCA-anonymity while also reducing both computational and communication costs. To support both forward security and non-frameability simultaneously, we introduce an updatable parameter into the user's credentials and public parameters so that newly joined users are unable to produce valid signatures for previously issued messages. This is because those messages are verifiable only under an expired public parameter. Finally, we propose a batch verification technique that aggregates pairing-based verification equations in our group signature scheme, enabling more efficient signature validation across multiple signatures.

3.2 First Protocol: DGSS

Let $DGSS = \{KeyGen, Join, Sign, Verify, Open, Judge, Update\}$ be our group signature scheme. In the following, we give the details of each procedure.

KeyGen. In this phase, a key generator generates public and private parameters of the scheme. Firstly, it creates $\{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2\}$ as the public parameter of the scheme where $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ ($\varphi(g_2) = g_1$) are two random generators. Subsequently, it generates a random value $z \in \mathbb{Z}_q$ as opening key (ok) and computes $h = g_1^z \in \mathbb{G}_1$. Additionally, it creates three secret values, b, β, and α as issuer keys (ik) and computes the public parameters $Y = g_1^{b\beta} \in \mathbb{G}_1$ and $w = g_2^{\alpha} \in \mathbb{G}_2$. Finally, the key generator securely send z and b, β, α to GM and key issuer, respectively. Furthermore, it selects a hash function $H : \{0,1\} \longrightarrow \mathbb{Z}_q$ and publishes the group public key of scheme as $gpk = \{g_1, g_2, h, Y, w, q, e, \mathbb{G}_1, \mathbb{G}_2, H(.)\}$.

Join. Whenever a user requests to join the group (assuming the request is valid), the key issuer generates a unique random value $a_i \in \mathbb{Z}_q$, computes $A_i = g_1^{\frac{b}{a_i}}$ and $x_i = \beta a_i + \alpha$, and securely transmits these values (x_i, A_i) to the user. Additionally, the key issuer records the pair (a_i, A_i) in the registration list as a valid user entry. The registration list is maintained in the form $Rlist = \{(a_1, A_1), (a_2, A_2), \ldots, (a_i, A_i)\}$.

Sign. To sign a message M, the signer performs the following steps:

1. The signer first generates a random value $k \in \mathbb{Z}_q$ and computes $T_1 = g_2^k, T_2 = A_i h^k$ and $T_3 = A_i^k$.
2. Then, the signer generates two random values r_k and r_e and evaluates $R_1 = g_2^{r_k}$ and $R_2 = e(A_i, g_2)^{r_e}$.
3. To prove the knowledge of $x_i = \beta a_i + \alpha$, A_i and k, the signer computes $c = H(M, T_1, T_2, T_3, R_1, R_2), s_k = r_k + ck$ and $s_e = r_e + ckx_i$. Finally, the signer sends $\sigma_i = \{T_1, T_2, T_3, c, s_k, s_e\}$ as the signature to the verifier.

Verify. In order to verify a signature σ_i, a verifier performs the following:

1. The verifier computes $\tilde{R_1} = g_2^{s_k} T_1^{-c}$ and $\tilde{R_2} = e(T_2, g_2)^{s_e} e(Y^{-c} h^{-s_e}, T_1) e(T_3^{-c}, w)$.
2. Then, it computes $c' = H(M, T_1, T_2, T_3, \tilde{R_1}, \tilde{R_2})$ and compares c' with c. If they are equals, the signature is valid, otherwise verification does not pass.

Open. This algorithm is run by GM and is used to trace a signature back to its signer. It takes as input a group public key $gpk = \{g_1, g_2, h, Y, w, e, \mathbb{G}_1, \mathbb{G}_2, H(.)\}$ and the corresponding GM private tracing key z, together with a message M and a signature $\sigma_i = (T_1, T_2, T_3, c, s_k, s_e)$. Then, it extracts the identity A_i of the signer. In order to reveal A_i, the GM (opener) firstly verifies that σ_i is a valid signature on M. then, it recovers the identity as $A_i \longleftarrow \frac{T_2}{\varphi(T_1)^z}$, then checks $e(A_i, T_1) \stackrel{?}{=} e(T_3, g_2)$. If so, it outputs A_i as the identity of signer, otherwise, it aborts. The open procedure requires isomorphism φ to reveal A_i. Computing the isomorphism takes roughly the same time as an exponentiation in the corresponding group (using fast computations of the trace map) [11].

Judge. Any entity can invoke the *Judge* algorithm to verify the opener's disclosure. This algorithm takes three inputs: a message M, a valid signature σ_i on M, and the A_i revealed by the opener. *Judge* first verifies the validity of

the signature σ_i. If the signature is valid, the *Judge* algorithm then validates the opening process by checking $e(A_i, g_2) \cdot e(h, T_1) \stackrel{?}{=} e(T_2, g_2)$. If successful, it outputs 1 otherwise it outputs 0.

Update. In the proposed protocol, each group member holds a secret key tuple (x_i, A_i), and the scheme's public parameters include $Y = g_1^{b\beta}$ and $w = g_2^{\alpha}$. To ensure forward security, we update the issuer's secret parameter b periodically, and accordingly, the group public parameters are also updated. The steps performed by the key issuer are as follows:

1. The key issuer broadcasts $l \in \mathbb{Z}_p$, $\delta = g_1^{\frac{b}{l}}$, and $\varkappa = \beta l - \alpha$ to all users where l is a random value.
2. Each user can then update their secret key component as $A_{i_{new}} = (\delta \cdot A_{i_{old}})^{\frac{1}{x_i + \varkappa}}$.
3. The key issuer publishes $Y_{new} = g_1^{b'\beta}$ as new public parameters where $b' = \frac{b}{\beta l}$.

Revocation. Our scheme enables the revocation of user membership without affecting the signing capability of other users.

1. The key issuer generates a random value $l \in \mathbb{Z}_p$ and computes $\delta = g_1^{\frac{b}{l}}$, $\varkappa = \beta l - \alpha$. To revoke a user with secret keys $x_i = \beta a_i + \alpha$ and A_i, the issuer broadcasts the values δ, $\varkappa$, x_i, and $A_i' = g_1^{\frac{b'}{a_i}}$ to all users, where $b' = \frac{b}{\beta l}$.
2. Upon receiving these transmissions, users update their secret identifiers by $A_{k_{new}} = (\frac{A_i'}{(\delta \cdot A_{k_{old}})^{\frac{1}{x_i + \varkappa}}})^{\frac{1}{(x_k - x_i)}}$ where $k \neq i$. Note that the revoked user is not able to update its $A_{i_{new}}$ because the new exponent is invalid.
3. Finally, the key issuer updates g_1 to $\hat{g}_1 = g_1^{\frac{1}{\beta a_i}}$ and publishes $Y_{new} = \hat{g}_1^{\beta b'}$ as a new public verification parameter.

In order to revoke user A_i, the key issuer first updates the user's credential from A_i to A_i' using the values δ and $\varkappa$. This step is necessary because, without updating A_i and publishing it, an adversary could link signatures previously generated by the revoked user. Specifically, the adversary could check whether $e(T_2, g_2) \stackrel{?}{=} e(A_i, g_2) \cdot e(h, T_1)$ holds for given values T_1 and T_2.

Note that we assume that an online or blockchain-based service is available to store the values $(\delta, \varkappa, x_i, A_i')$, which users must check regularly to update their keys. Otherwise, their signatures are not verifiable.

3.3 Second Protocol: xDGSS

In xDGSS, we build on DGSS to present a dynamic group signature scheme that ensures non-frameability. Instead of simply assigning user i the private key $A_i = g_1^{\frac{b}{a_i}}, x_i = \beta a_i + \alpha$, the user and key issuer engage in an interactive protocol such that key issuer is not able to know the user's secret signing. Note that *Open* and *Judge* algorithms are the same as in DGSS.

KeyGen. In addition to public values produced in previous protocol, the key generator adds two extra public values as $h_1 \in \mathbb{G}_1$ and $Y_1 = h_1^{b\beta} \in \mathbb{G}_1$.
Join: User i generates a random number $y_i \in \mathbb{Z}_q$, computes $L_i = h_1^{-y_i}$ and sends L_i to the key issuer via secure channel. The key issuer then generates a random number a_i and evaluates $A_i = (g_1 L_i)^{\frac{b}{a_i}} = (g_1 h_1^{-y_i})^{\frac{b}{a_i}}$ and sends the pair $(A_i, x_i = \beta a_i + \alpha)$ to the user.
Sign. The signing procedure in the second protocol follows the same steps as in the first, with the following modifications. The value R_2 is updated to $R_2 = e(A_i, g_2)^{r_e} \cdot e(Y_1, g_2)^{k r_{y_i}}$, and an additional response value $s_y = r_y + c y_i$ is computed, where r_y is a newly chosen random value. The final signature consists of $T_1, T_2, T_3, c, s_k, s_e, s_y$.
Verify. The verification process is the same as in the first protocol, except that $\tilde{R}_2$ is computed as $\tilde{R}_2 = e(T_2^{s_e}, g_2) \cdot e(h^{-s_e} Y^{-c} Y_1^{s_y}, T_1) \cdot e(T_3^{-c}, w)$. The signature is accepted if $c' = c$, and rejected otherwise.
Update. Similar to DGSS, the key issuer periodically updates the public parameters and helps users refresh their secret keys. In addition to the steps in the original scheme, the issuer now also broadcasts $\gamma = h_1^{\frac{b}{l}}$ and publishes an extra public value $Y_{1_{\text{new}}} = h_1^{b'\beta}$. Each user updates their secret key component as $A_{i_{\text{new}}} = (\delta \cdot \gamma^{y_i} \cdot A_{i_{\text{old}}})^{\frac{1}{x_i + \varkappa}}$.

This ensures that new members, whose keys align with the updated parameters $Y_{\text{new}} = g_1^{b'\beta}$ and $Y_{1_{\text{new}}} = h_1^{b'\beta}$, cannot forge valid signatures for messages verified under the previous public keys $Y = g_1^{b\beta}$ and $Y_1 = h_1^{b\beta}$.
Revocation. Similar to the first revocation phase, the key issuer broadcasts $x_i = \beta a_i + \alpha$ of the revoked user. Additionally, the issuer sends $s = h_1^{-\frac{b}{a_i}}$ and $s' = g_1^{\frac{b}{a_i}}$. Each remaining user computes $X = s' \cdot s^{y_k}$ and updates their secret key as $A_{k_{\text{new}}} = \left(\frac{X}{A_{k_{\text{old}}}}\right)^{\frac{1}{x_k - x_i}}$ for $k \neq i$. Since the revoked user cannot compute a valid exponent due to division by zero, it loses the ability to generate updated credentials. The key issuer then updates the public keys: g_1 and h_1 are changed to $\hat{g}_1 = g_1^{\frac{1}{\beta a_i}}$, $\hat{h}_1 = h_1^{\frac{1}{\beta a_i}}$, and publishes the new verification parameters $Y_{new} = \hat{g}_1^{\beta b}$ and $Y_{1_{new}} = \hat{h}_1^{\beta b}$.

3.4 Batch Verification

To improve efficiency in applications verifying multiple group signatures, we modify our protocol to support batch verification, minimizing costly pairing operations on the verifier side. Instead of computing R_2, the signer computes $T_4 = A_i^{r_e}$ and includes it in the hash: $c = H(M, T_1, T_2, T_3, T_4, R_1)$. The final signature σ includes $T_1, T_2, T_3, T_4, c, s_k, s_e, s_y$. On the verifier's side, the verification process involves computing $\tilde{R}_1$ and $c' = H(M, T_1, T_2, T_3, T_4, \tilde{R}_1)$. The verifier first checks $c \stackrel{?}{=} c'$ for each received signature and then verifies the fol-

lowing equalition:

$$e(\prod_{i=1}^{t} T_{4_i}, g_2)e(h, \prod_{i=1}^{t} T_{1_i}^{s_{e_i}})e(Y, \prod_{i=1}^{t} T_{1_i}^{c_i})e(\prod_{i=1}^{t} T_{3_i}^{c_i}, w) \stackrel{?}{=} e(\prod_{i=1}^{t} T_{2_i}^{s_{e_i}}, g_2) \quad (1)$$

If both equalities hold, the verifier confirms the validity of the signer.

For a group signature scheme with non-frameability, a similar approach can be applied. In this case, the signers compute $T_4 = A_i^{r_e} g_2^{r_{y_i}}$ and equation 1 is modified to:

$$e(\prod_{i=1}^{t} T_{4_i}, g_2)e(h, \prod_{i=1}^{t} T_{1_i}^{s_{e_i}})e(Y, \prod_{i=1}^{t} T_{1_i}^{c_i})e(Y_1, \prod_{i=1}^{t} T_1^{-s_{y_i}})e(\prod_{i=1}^{t} T_{3_i}^{c_i}, w) \stackrel{?}{=} e(\prod_{i=1}^{t} T_{2_i}^{s_{e_i}}, g_2)$$

The soundness of the NIZK proof in the DGSS scheme with batch verification is provided in Appendix A. For the xDGSS scheme, the soundness analysis remains identical to that of DGSS.

3.5 Zero Knowledge Proof of the Protocol

Our DGSS is built on a zero-knowledge proof (ZKP) by which the signer convinces the verifier that it possesses valid values x_i and A_i such that $e(A_i, g_2^{x_i} w^{-1}) = e(Y, g_2)$ and that A_i is encrypted using public key h in T_1 and T_2 and the secret value k. A secure ZKP should ensure *completeness* which implies that if the statement is true, an honest signer can convince a verifier of that fact. It should also ensure *soundness* which means that if the statement is false, no cheating signer can convince the verifier that it is true, except with some small probability. ZKPs achieves *zero-knowledge* which implies that after verification, the verifier learns nothing about the witness other than the fact that the verification is true [21, 8]. In the protocols, we apply the Fiat-Shamir heuristic [20] to convert an interactive zero-knowledge scheme into a non-interactive one. For the purpose of our proof, we assume the protocol is non-interactive.

In the following, we explain how DGSS satisfies the zero-knowledge properties, while the proof for the xDGSS scheme follows straightforwardly.

Lemma 1 *The NIZK in the DGSS protocol is complete.*

Proof. If the signer is an honest prover possessing a valid pair (x_i, A_i), then they can generate a valid signature $\sigma_i = (T_1, T_2, T_3, c, s_k, s_e)$. Upon receiving the signature, the verifier computes $\tilde{R}_1$ and $\tilde{R}_2$. If the signature is valid, these values satisfy $\tilde{R}_1 = R_1$ and $\tilde{R}_2 = R_2$. Consequently, the verifier accepts $c = c'$.

Lemma 2 *There exists an extractor for the NIZK in DGSS protocol.*

Proof. We assume an interactive ZKP where c is a challenge sent by the verifier after the prover sends the values T_1, T_2, T_3, R_1, R_2 to the verifier. Assume a scenario where an extractor can rewind a prover in the protocol to just before

the prover receives a challenge c. In the initial step of the protocol, the prover sends T_1, T_2, T_3 along with R_1, R_2. When faced with the challenge c, the prover responds with s_k and s_e. For a different challenge $c' \neq c$, the prover responds with s'_k and s'_e. Hence, the following two verification equations hold for each set of values.

$$g_2^{s_k} \stackrel{?}{=} R_1 T_1^c$$
$$e(T_2, g_2)^{s_e} \stackrel{?}{=} R_2 e(h^{s_e} Y^c, T_1) e(T_3^c, w)$$

Therefore,

$$g_2^{s_k} = R_1 T_1^c, g_2^{s'_k} = R_1 T_1^{c'} \quad (2)$$
$$e(T_2, g_2)^{s_e} = R_2 e(h^{s_e} Y^c, T_1) e(T_3^c, w),$$
$$e(T_2, g_2)^{s'_e} = R_2 e(h^{s'_e} Y^{c'}, T_1) e(T_3^{c'}, w) \quad (3)$$

We define three new parameters such that $\Delta c = c - c', \Delta s_k = s_k - s'_k$ and $\Delta s_e = s_e - s'_e$.

By dividing the two instances of Eq. 2 (one instance using c and the other using c'), we obtain $g_2^{\Delta s_k} = T_1^{\Delta c}$ meaning that $T_1 = g_2^{\frac{\Delta s_k}{\Delta c}}$. Therefore, we reveal the value of k as $k = \frac{\Delta s_k}{\Delta c}$. Similarly, the value k can be obtained from the equations $s_k = r_k + ck$ and $s'_k = r_k + c'k$. By subtracting the equations, we get $\Delta s_k = \Delta c \cdot k$, which leads to $k = \frac{\Delta s_k}{\Delta c}$.

On the other hand, by dividing the two instance of Eq. 3, the following holds.

$$e(A_i, g_2)^{\frac{\Delta s_e}{\Delta c}} = e(Y, T_1) e(T_3, w)$$

Therefore, $kx_i = \frac{\Delta s_e}{\Delta c}$. In addition, we have $k = \frac{\Delta s_k}{\Delta c}$, therefore $x_i = \frac{\Delta s_e}{\Delta s_k}$. Similarly, the value x_i can be computed from the equations $s_e = r_e + cx_i k$ and $s'_e = r_e + c'x_i k$. By subtracting the equations, we get $\Delta s_e = \Delta c k x_i$, which leads to $x_i = \frac{\Delta s_e}{\Delta s_k}$.

Finally, one can obtain A_i from $T_2 = A_i h^k$. Hence, an extractor exists for (x_i, A_i, k), therefore the ZKP is sound.

Lemma 3 *Transcripts of the DGSS protocol can be simulated.*

Proof. : We describe a simulator that outputs transcripts of the protocol. The simulator begins by picking $A \in \mathbb{G}_1$ and $k \in \mathbb{Z}_q$. It sets $T_1 = g_2^k$, $T_2 = Ah^k$, and $T_3 = A^k$. Since the verifier does not know the private key of the ElGamal encryption, it cannot recognize T_1, T_2 and T_3 are generated via invalid A. Now, the simulator computes other values without knowing A_i, x_i, k and constructs the transcript perfectly, following the standard simulation of a Schnorr proof of knowledge [33]. The simulator chooses a challenge $c \in \mathbb{Z}_q$ and values s_k, s_e. It computes R_1, R_2 as follows:

$$R_1 = g_2^{s_k} T_1^{-c},\ R_2 = e(T_2, g_2)^{s_e} e(Y^{-c} h^{s_e}, T_1) e(T_3, w)^{-c}$$

The simulator outputs the transcript $(T_1, T_2, T_3, R_1, R_2, c, s_k, s_e)$. This transcript is verifiable since $\tilde{R}_1$ and $\tilde{R}_2$ are computed in the same manner. Moreover, it is indistinguishable from the transcripts generated by the protocol.

4 Security Analysis

In this section, we demonstrate that our DGSS scheme satisfies the security requirements of anonymity, traceability and forward secrecy. Additionally, we show that xDGSS also satisfies non-frameability. Since the traceability, anonymity and forward secrecy properties of xDGSS are similar to those of the DGSS scheme, we leave their verification to the reader.

Theorem 1. *If the CDH problem is intractable in G_1, then the group signature scheme is anonymous.*

Proof. Given an anonymity adversary $\mathcal{A}$ that takes as input the public parameters of the group signature, then queries its challenger with i_0 and i_1, corresponding to identifiers A_0 and A_1, and receives a group signature σ_b by i_b where b is a secret random bit. $\mathcal{A}$ wins if it returns bit $b' = b$, i.e., recognizes the identifier of the signer (A_b). We will show how to use this adversary to create an adversary $\mathcal{B}$ to solve the CDH.

The CDH challenger randomly generates $x, p \in \mathbb{Z}_q$ and computes $X = g_1^x$ and $P = g_1^p$. They are given to $\mathcal{B}$, and $\mathcal{B}$ is required to output the value $Z = g_1^{xp}$. $\mathcal{B}$ simulates xDGSS to $\mathcal{A}$, it sets $h = X$ and $g_2 = P^{\frac{1}{\theta}}$ where θ is randomly selected such that $P^{\frac{1}{\theta}} \in \mathbb{G}_2$. $\mathcal{B}$ then computes $w = P^{\alpha}$ and $Y = g_1^{f\theta}$ where f and α are security parameter of group signature and are selected randomly from Z_p. Finally, it sends the public values $(g_1, g_2 = P^{\frac{1}{\theta}}, h = X, Y = g_1^{f\theta}, w = P^{\alpha})$ along with the users' private keys (except target users) $\{(A_i = g_1^{\frac{f}{\alpha_i}}, x_i = \theta(\beta a_i + \alpha))\}_{i=1}^{n}$ to $\mathcal{A}$ which queries its anonymity challenge by providing i_0, i_1 and the message to be signed M. $\mathcal{B}$ selects one of users $b \in \{0, 1\}$ randomly, generates a random value $s \in \mathbb{G}_1$ and calculates the signature to pass to $\mathcal{A}$ as follows:

$$T_1 = P \in \mathbb{G}_2, T_2 = sh^{\theta}, T_3 = s^{\frac{\beta a_b + \alpha}{\alpha\theta}} g_1^{-\frac{f\beta}{\alpha}}, R_1 = P^{r_k}, R_2 = e(s, g_2)^{r_e},$$
$$c = H(M, T_1, T_2, T_3, R_1, R_2), s_k = \theta(r_k + c), s_e = r_e + c(\beta a_b + \alpha)$$

When $\mathcal{A}$ sends back b' as its response, if $b \neq b'$, the adversary $\mathcal{B}$ aborts, otherwise, by dividing T_2 by A_b, the algorithm $\mathcal{B}$ computes the value h^p. This follows from the assumption that T_2 is of the form $A_b h^p$; indeed, since $sh^{\theta} = A_b h^p$, it implies that $\mathcal{B}$ successfully recovers h^p. Consequently, $\mathcal{B}$ can compute $Z = g_1^{xp}$. However, this contradicts the assumed hardness of the Computational Diffie-Hellman (CDH) problem. Therefore, such an adversary $\mathcal{B}$ cannot exist, which in turn implies that the adversary $\mathcal{A}$ cannot exist either.

Theorem 2. *The proposed group signature scheme is traceable under SDH assumption.*

Proof. Suppose $\mathcal{A}$ breaks the traceability of the group signature scheme. We show how to construct an adversary $\mathcal{B}$ that breaks the intractability of the SDH assumption.

$\mathcal{B}$ is challenged with an $n+1$-SDH instance $\{g, \tilde{g}, \tilde{g}^{\theta}, \tilde{g}^{\theta^2}, \ldots, \tilde{g}^{\theta^{n+1}}\}$. It simulates the DGSS protocol to $\mathcal{A}$ by randomly choosing $b, \beta \leftarrow \mathbb{Z}_q$ and $a_i \leftarrow \mathbb{Z}_q$

for $i = 1, \ldots, n$. It defines $g_2 = \tilde{g}^{\prod_{i=1}^{n}(\beta a_i+\theta)} = \tilde{g}^{\sum_{i=1}^{n}(\nu_i\theta^i)}$, and $g_1 = \varphi(g_2)$. For each user, the adversary $\mathcal{B}$ computes a pair of signing keys as $x_i = \beta a_i$ and $A_i = g^{b\prod_{j=1,j\neq i}^{n}(\beta a_j+\theta)}$. The point is that a_i is uniquely used across all join protocol executions. This ensures that A_i and x_i are also unique.

$\mathcal{B}$ considers $\alpha = -\theta$, which is unknown, and computes w as $g_2 = \tilde{g}^{-\theta\prod_{i=1}^{n}(\beta a_i+\theta)} = \tilde{g}^{\sum_{i=1}^{n}(\zeta_i\theta^i)}$.

$\mathcal{B}$ computes the public value Y as follows:

$$Y = \varphi\left(\tilde{g}^{b\prod_{i=1}^{n}(\beta a_i+\theta)}\right) = g^{b\prod_{i=1}^{n}(\beta a_i+\theta)}. \tag{4}$$

It then picks a random value $z \in \mathbb{Z}_q$ and computes $h = g_1^z \in \mathbb{G}_1$. $\mathcal{B}$ initializes $\mathcal{A}$ by providing it with the group public key (g_1, g_2, h, w, Y) and the secret opening key z.

The $\mathcal{A}$ outputs a forged group signature $\sigma^* = (T_1^*, T_2^*, T_3^*, c^*, s_k^*, s_e^*)$ on a message M^*.

There are two types of traceability adversaries; a valid signature that the opener fails to identify its signer (*Type I*) or a valid signature that is traced back to an honest user via the open and judge algorithms (*Type II*).

Type I. If the adversary $\mathcal{A}$ generates a signature σ^*, $\mathcal{B}$ can calculate A_i^* using the opening key ok and can also compute $x_i^* = \beta a_{i^*} + \alpha$ using extractor $\mathcal{E}$.

Consequently, $\mathcal{B}$ can compute $(A_i^*)^{(b\beta)^{-1}} = \left(g_1^{\frac{b}{\alpha_i^*}}\right)^{(b\beta)^{-1}} = g_1^{\frac{1}{x_i^*+\theta}}$. On the other hand, observe that $g_1^{\frac{1}{x_i^*+\theta}} = \varphi(g_2)^{\frac{1}{x_i^*+\theta}} = \varphi(\tilde{g}^{f(\theta)})^{\frac{1}{x_i^*+\theta}} = g^{\frac{f(\theta)}{x_i^*+\theta}}$, where $f(\theta) = \prod_{i=1}^{n}(\beta a_i+\theta)$. It is clear that $f(\theta)$ can be expressed as $f(\theta) = \sum_{i=0}^{n}\eta_i\theta^i$, with $\eta_i \in \mathbb{Z}_q$. Using polynomial long division, we can write $f(\theta) = g(\theta)(x_i^* + \theta) + k$, where $g(\theta) = \sum_{i=1}^{n-1}\eta_i\theta^i$ and $k \in \mathbb{Z}_q$ is the remainder. Therefore,

$$g_1^{\frac{1}{x_i^*+\theta}} = g^{\frac{k}{x_i^*+\theta}+g(\theta)} = g^{\frac{k}{x_i^*+\theta}+\sum_{i=1}^{n-1}\eta_i\theta^i}.$$

Thus, the adversary $\mathcal{B}$ can compute $g^{\frac{1}{x_i^*+\theta}}$ using $(g_1^{\frac{1}{x_i^*+\theta}} \cdot \prod_{i=1}^{n-1} g^{(\theta^i)^{-\eta_i}})^{k^{-1}}$ because it knows the coefficients of $f(\theta)$.

Finally, it sends $(g^{\frac{1}{x_i^*+\theta}}, x_i^*)$ to its challenger as a solution for the SDH problem.

Type II. In this case, the adversary $\mathcal{A}$ generates a signature such that the GM traces it back to an honest member with identifier A_j. In addition, the returned value from GM, A_j, passes the *Judge* algorithm, which implies $e(A_j, g_2)\cdot e(h, T_1) = e(A^*, g_2)e(h^k, g_2)$.

It then means that $e(A_j, g_2) = e(h^k, g_2)$ (or $e(A^*, g_2) = e(h, T_1)$) for some $A^* \neq A_j$, then $A_j = h^k$. In that case, the $\mathcal{B}$ can compute $g_1^{\frac{1}{x_j+\theta}}$ via $\varphi(T_1^{z(b\beta)^{-1}})$. Consequently, $\mathcal{B}$ can solve the SDH problem such as the previous case. Since the SDH problem is hard, $\mathcal{B}$ does not exist and therefore, $\mathcal{A}$ cannot exist.

Theorem 3. *Assuming that the zero-knowledge proof is simulatable and extractable, xDGSS group signature is non-frameable under the discrete logarithm problem.*

Proof (*sketch*). Let $Y = g_1^y$ be an instance of the discrete logarithm (DL) problem, and let $\mathcal{S}$ and $\mathcal{E}$ be the zero-knowledge simulator and the knowledge extractor, respectively. The proof idea is that the DL adversary $\mathcal{B}$ assumes the unknown $y_i = \log_{g_1} Y$ as a signing key of the framing target user i^*. When $\mathcal{A}$ succeeds in forging a signature with respect to y_i, $\mathcal{B}$ can solve the DL problem (using $\mathcal{E}$). In this game, the adversary $\mathcal{A}$ can act as the key issuer during the join phase and generate a secret key x_i and a credential A_i in response to a join request. $\mathcal{B}$ randomly generates $g_1, g_2, \alpha, \beta, b, z$ and computes $w = g_2^{\alpha}, Y = g_1^{b\beta}$. Finally, $\mathcal{B}$ transfers issuer keys, opener key and public parameters to $\mathcal{A}$ which is allowed to request signing keys of any identity i as follows:

- If $i \neq i^*$, $\mathcal{B}$ generates y_i and computes $L_i = g_1^{-y_i}$ and sends to $\mathcal{A}$ which sends back partial secret parameters (x_i, A_i) to $\mathcal{B}$ using the join procedure.
- If $i = i^*$, $\mathcal{B}$ sets $L_{i^*} = Y$ and sends it to $\mathcal{A}$ which sends back $(x_i, A_i = (g_1 X)^{\frac{b}{a_{i^*}}})$. $\mathcal{B}$ then stores whole secret parameters as (y_i, x_i, A_i) if $i \neq i^*$; otherwise $(\perp, x_{i^*}, A_{i^*})$.

To respond to signing queries using identity i and a message M, if $i \neq i^*$, $\mathcal{B}$ is able to generate a group signature on any message, because $\mathcal{B}$ has the corresponding signing key. On the other hand, if $i = i^*$, $\mathcal{B}$ can produce a signature using the stored partial secret credentials and the NIZK simulator $\mathcal{S}$. The resulting signature σ is then given to $\mathcal{A}$.

At the end, $\mathcal{A}$ returns M^*, $\sigma^* = (T_1^*, T_2^*, T_3^*, c^*, s_k^*, s_e^*, s_y^*)$ and A_i that passes *Judge* algorithm. This indicates that since $L = Y$, the exponent $y = \log_{g_1} X$ is a NIZK witness. In that case, $\mathcal{B}$ can obtain y using extractor $\mathcal{E}$, which is the solution of the DL problem. Since the DL problem is hard, $\mathcal{B}$ does not exist and therefore, $\mathcal{A}$ cannot exist.

Theorem 4. *The proposed group signature scheme is forward secure under SDH assumption.*

Proof. (*sketch*) Suppose an adversary $\mathcal{A}$ breaks the forward secrecy of the group signature scheme by producing a valid signature $\sigma^* = (T_1^*, T_2^*, T_3^*, c^*, s_k^*, s_e^*)$ that verifies under the old group public key $Y_{\text{old}} = g_1^{b\beta}$, despite not possessing the secret key corresponding to the previous user public key $A_i = g_1^{\frac{b}{a_i}}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that solves the $(n+1)$-SDH problem.
Given an $(n+1)$-SDH instance $\{g, \tilde{g}, \tilde{g}^{\theta}, \tilde{g}^{\theta^2}, \ldots, \tilde{g}^{\theta^{n+1}}\}$, $\mathcal{B}$ simulates the DGSS protocol for $\mathcal{A}$ by choosing $b, b', \beta \leftarrow \mathbb{Z}_q$ and $a_i \leftarrow \mathbb{Z}_q$ for $i = 1, \ldots, n$. It sets $g_2 = \tilde{g}^{\prod_{i=1}^{n}(\beta a_i + \theta)} = \tilde{g}^{\sum_{i=1}^{n} \nu_i \theta^i}$ and $g_1 = \varphi(g_2)$. For each user, $\mathcal{B}$ computes $x_i = \beta a_i$ and $A_i = g^{b'} \prod_{j=1, j\neq i}^{n} (\beta a_j + \theta)$. Each a_i is unique, ensuring unique x_i and A_i. Letting $\alpha = -\theta$ (unknown to $\mathcal{B}$), it defines $g_2 = \tilde{g}^{-\theta \prod_{i=1}^{n}(\beta a_i + \theta)} = \tilde{g}^{\sum_{i=1}^{n+1} \zeta_i \theta^i}$, and computes $Y_{old} = g^{b \prod_{i=1}^{n}(\beta a_i + \theta)}$ and $Y_{new} = g^{b' \prod_{i=1}^{n}(\beta a_i + \theta)}$. $\mathcal{A}$ outputs a forged group signature $\sigma^* = (T_1^*, T_2^*, T_3^*, c^*, s_k^*, s_e^*)$ on message M^* verified by Y_{old}. Using the opening key ok, $\mathcal{B}$ extracts A_i^* and $x_i = \beta a_i + \alpha$, then computes $(A_i^*)^{(b\beta)^{-1}} = g_1^{1/(x_i^* + \theta)}$, which gives $g^{1/(x_i^* + \theta)}$ as described in Theorem 2. Thus, $\mathcal{B}$ solves the SDH problem. Since SDH is assumed hard, $\mathcal{B}$ cannot exist, and therefore, neither can $\mathcal{A}$.

5 Performance and Comparison

In this section, we compare our protocol with others in terms of security properties. Additionally, we implement the basic cryptographic operations of the protocol, measure their execution time and and compare our protocol with other schemes in terms of computational cost, communication overhead, and signature size.

5.1 Security and Feature Comparison

In this subsection, we compare group signature schemes based on their security properties and functionalities. Table 1 summarizes key features across schemes, identified by authors (e.g., BBS for Boneh, Boyen, and Shacham [10]). Notations used are clarified below.

For anonymity, CCA2-anonymity (CCA2) is the strongest, allowing both opening and signing key queries, except on the challenge pair. CPA-anonymity (CPA) [10] disallows opening queries, while CCA-anonymity (CCA) forbids signing key queries [18].

Two related notions, non-frameability and strong exculpability (SE), differ slightly: non-frameability prevents any entity, including the issuer, from forging signatures accepted by both the opening and judging algorithms; SE only ensures that no valid signature can be forged on behalf of a user. Since non-frameability implies SE, we use SE for broader comparison.

Some schemes suffer from inefficient opening algorithms with $\mathcal{O}(n)$ complexity, unsuitable for large groups. Efficient schemes support constant-time opening $\mathcal{O}(1)$. Batch verification is also critical, allowing the verifier to process multiple signatures with a single algorithm run, reducing load in centralized systems.

Forward security prevents new members from signing past messages. Our scheme enables efficient key updates by broadcasting just two values, avoiding full re-enrollment. Revocation is another essential property, allowing removal of compromised users. Schemes supporting both revocation and forward security offer stronger guarantees.

In Table 1, we compare protocols on key properties. Only a few, such as BBS^2 [17], LMP [26], CS [16], DS [18], ACJT [2], and our scheme, provide CCA2-anonymity. Among them, DP [17], ACJT [2], and our scheme also support practical revocation and efficient opening.

While several protocols support SE, batch verification is less common, only KLA [24] and CL [12] provide it. However, these lack efficient key update mechanisms. Our scheme uniquely combines forward security with efficient key updates, a feature absent in prior work. We analyze the computational costs of these properties in the next subsection.

5.2 Computation and Communication Performance

To evaluate our schemes in comparison with other related protocols, we group the analysis based on shared properties. Therefore, the efficiency of our work

Scheme	Anonymity	Revocation	Opening	BV	SE	EFS
BBS [10]	CPA	✓	$\mathcal{O}(1)$	✗	✗	✓
BBS^{+} [10]	CPA	✗	$\mathcal{O}(1)$	✗	✓	✗
BS [11]	CPA	∇	$\mathcal{O}(1)$	✗	✗	✗
$FGHP$ [19]	CCA2	✗	$\mathcal{O}(1)$	✓	✗	✗
LMP [26]	CCA2	✗	$\mathcal{O}(1)$	✗	✗	✗
PS [32]	CCA	✗	$\mathcal{O}(n)$	✓	✗	✗
DS [18]	CCA2	✗	$\mathcal{O}(n)$	✗	✗	✗
CS [16]	CCA2	✗	$\mathcal{O}(n)$	✗	✗	✗
BBS^{++}[9]	CPA	✗	$\mathcal{O}(1)$	✗	✓	✗
$BSNS$ [9]	CCA	∇	$\mathcal{O}(n)$	✗	✓	✗
CL [12]	CCA	✗	$\mathcal{O}(n)$	✓	✓	✗
DP [17]	CCA2	✓	$\mathcal{O}(1)$	✗	✓	✗
$ACJT$ [2]	CCA2	✓	$\mathcal{O}(1)$	✗	✓	✗
$SKHS$ [36]	CPA	✓	$\mathcal{O}(1)$	✗	✓	✗
KLA [24]	CCA	✗	$\mathcal{O}(n)$	✓	✓	✗
Our DGSS	CCA2	✓	$\mathcal{O}(1)$	✓	✗	✓
Our xDGSS	CCA2	✓	$\mathcal{O}(1)$	✓	✓	✓

Table 1: Security and property comparison between different schemes.
BV: Batch Verification. SE: Strong Exculpability (non-frameability). EFS: Efficient Forward Security. ∇: This scheme does not have revocation protocol, but it has a revocation list.

is assessed in two distinct classes; schemes with strong exculpability and those without strong exculpability.

To compute the communication and computation overheads, we adopt the Barreto-Naehrig (BN254) curve over $\mathbb{F}_p$ where p is a 254-bit prime number [5]. For computational evaluation, we focus exclusively on expensive operations, specifically exponentiation in $\mathbb{G}_T$, scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$, and pairing operations.

To evaluate computational performance across various scenarios, we conducted measurements on a laptop equipped with a Core i7 CPU (2.6 GHz) and 8 GB of RAM. The results indicate that a single exponentiation operation in $\mathbb{G}_T$, denoted by E_T, requires approximately 0.14 ms. Additionally, scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$ take around 0.76 ms. A pairing operation, represented by P, requires approximately 4.45 ms. These times are calculated by averaging the execution duration of each operation over 100,000 runs.
Schemes without strong exculpability. In DGSS, which does not provide non-frameability, the signer outputs a signature $(T_1, T_2, T_3, c, s_k, s_e)$, totaling 192 bytes assuming a 32-byte hash function. Table 2 compares group signature schemes without strong exculpability in terms of communication and computation. While CS [16] and PS [32] offer smaller signature sizes than DGSS, they lack efficient opening and revocation mechanisms, and require re-running the join

phase to update signing keys for forward security. Among comparable schemes, DGSS achieves the best communication efficiency.

In terms of signer-side computation, DGSS is more efficient than most schemes, with the exception of PS [32], which lacks comparable security guarantees. On the verifier side, BBS [10], PS [32], and FGHP [19] exhibit better performance than DGSS. However, only PS supports batch verification, whereas DGSS includes this feature, making it more suitable for scalable applications.

Schemes	Signature size (Bytes)	Signer (ms)	Verifier(ms)
BBS [10]	$3\mathbb{G}_1 + 6\mathbb{Z}_q$ (288)	$9M + 3E_T$ (7.26)	$10M + 3E_T + P$ (12.47)
BS [11]	$2\mathbb{G}_1 + 5\mathbb{Z}_q$ (224)	$10M + 2P$ (16.5)	$10M + 3P$ (20.95)
$FGHP$ [19]	$3\mathbb{G}_1 + 6\mathbb{Z}_q + \mathbb{G}_T$ (672)	$9M + 3E_T$ (7.26)	$10M + 3E_T + P$ (12.47)
LMP [26]	$7\mathbb{G}_1 + 3\mathbb{Z}_q$ (320)	$13M + 2E_T + 4P$ (27.96)	$15M + 4P$ (29.2)
PS [32]	$2\mathbb{G}_1 + 2\mathbb{Z}_q$ (128)	$2M + E_T$ (1.66)	$3M + 3P$ (15.63)
DS [18]	$4\mathbb{G}_2 + 3\mathbb{G}_1 + 3\mathbb{Z}_q$ (320)	$11M$ (8.36)	$6M + 5P$ (26.81)
CS [16]	$\mathbb{G}_2 + 4\mathbb{G}_1$(160)	$6M$ (4.56)	$M + 5P$ (23.01)
Our DGSS	$2\mathbb{G}_1 + \mathbb{G}_2 + 3\mathbb{Z}_q$(192)	$4M + E_T$ (3.18)	$6M + 3P$ (17.91)

Table 2: Comparison of group signature without strong exculpability

Schemes with strong exculpability. xDGSS ensures non-frameability by including one additional element (s_y) in the signature compared to DGSS, increasing the total size by 32 bytes to 224 bytes. It also requires one extra point multiplication on both the signer and verifier sides. As shown in Table 3, the schemes in [9] and [24] achieve lower communication costs than xDGSS. However, both suffer from expensive opening phases for large groups, as the group manager must perform exhaustive searches to identify the signer. Moreover, the scheme in [24] lacks revocation support, and [9] incurs significant overhead during revocation. In contrast, xDGSS enables efficient key updates by broadcasting only two values, avoiding the need to re-initiate the join phase for all users.

xDGSS also demonstrates strong signer-side efficiency, outperforming most existing schemes except [9], which lacks comparable features. This makes xDGSS particularly suitable for resource-constrained environments such as IoT applications. On the verifier side, schemes like BBS [10], BBS^3 [9], CL [12], BSNS [9], ACJT [2], and KLA [24] offer lower computational costs. However, batch verification, a critical feature for scalability, is only supported by xDGSS and CL [12]. For a fair comparison, we assume ACJT [2] operates over $\mathbb{Z}_q$ with a 2048-bit prime to match the 127-bit security level.

6 Conclusion

We proposed two forward-secure dynamic group signature schemes that balance efficiency and security. The first scheme, DGSS, prioritizes efficiency but does

Schemes	Signature size (Bytes)	Signer (ms)	Verifier(ms)
BBS^{+} [10]	$3\mathbb{G}_1 + 7\mathbb{Z}_q$ (320)	$9M + 4E_T$ (7.4)	$10M + 4E_T + P$ (12.61)
BBS^{++}[9]	$4\mathbb{G}_1 + 5\mathbb{Z}_q$ (288)	$11M + 5E_T$ (9.06)	$15M + 2P$ (20.3)
$BSNS$ [9]	$3\mathbb{G}_1 + 2\mathbb{Z}_q$ (160)	$3M + E_T$ (2.42)	$3M + 4P$ (20.08)
CL [12]	$7\mathbb{G}_1 + 4\mathbb{Z}_q$ (352)	$13M + 2E_T$ (10.16)	$11M + 4P$ (26.16)
DP [17]	$4\mathbb{G}_1 + 5\mathbb{Z}_q$ (288)	$7M + 4E_T$ (5.88)	$12M + 2P$ (18.02)
$ACJT$ [2]	$3\mathbb{G}_1 + 4\mathbb{Z}_q$ (1792)	$12E_T$ (1.68)	$11E_T$ (1.54)
$SKHS$ [36]	$6\mathbb{G}_1 + 6\mathbb{Z}_q$ (384)	$15M + 6P$ (38.1)	$7M + 6P$ (32.02)
KLA [24]	$3\mathbb{G}_1 + 2Z_p$ (160)	$4M$ (3.04)	$2M + 3P$ (14.87)
Our xDGSS	$2\mathbb{G}_1 + \mathbb{G}_2 + 4\mathbb{Z}_q$ (224)	$4M + 2E_T$ (3.32)	$7M + 3P$ (18.67)

Table 3: Comparison of group signature without strong exculpability

not ensure non-frameability, while the second scheme, xDGSS, provides non-frameability with minimal additional computational and communication cost. Both schemes introduce an efficient revocation mechanism and a batch verification technique that reduces verification overhead. Our key update mechanism allows for forward security without requiring all users to rejoin the group, overcoming a major limitation in existing solutions. Through security analysis, we demonstrated that our schemes satisfy key security properties, including anonymity, traceability, and non-frameability (for the second scheme). Performance evaluations indicate that our protocols are compatitive compared to existing group signature schemes, particularly in signature generation and batch verification.

References

1. AlTawy, R., Gong, G.: Mesh: A supply chain solution with locally private blockchain transactions. p. 149–169 (2019)
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) Advances in Cryptology — CRYPTO 2000. pp. 255–270. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
3. Ateniese, G., Song, D., Tsudik, G.: Quasi-efficient revocation of group signatures. In: Blaze, M. (ed.) Financial Cryptography. pp. 183–197. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
4. Ateniese, G., Tsudik, G.: Some open issues and new directions in group signatures. In: Franklin, M. (ed.) Financial Cryptography. pp. 196–211. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
5. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) Selected Areas in Cryptography. pp. 319–331. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
6. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) Advances in Cryptology — EUROCRYPT 2003. pp. 614–629. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)

7. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: Menezes, A. (ed.) Topics in Cryptology – CT-RSA 2005. pp. 136–153. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
8. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: Wang, X., Sako, K. (eds.) Advances in Cryptology – ASIACRYPT 2012. pp. 626–643. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
9. Bichsel, P., Camenisch, J., Neven, G., Smart, N.P., Warinschi, B.: Get shorty via group signatures without encryption. In: Garay, J.A., De Prisco, R. (eds.) Security and Cryptography for Networks. pp. 381–398. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
10. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. pp. 41–55. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
11. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. p. 168–177. CCS '04, Association for Computing Machinery, New York, NY, USA (2004)
12. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. pp. 56–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
13. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) Advances in Cryptology — EUROCRYPT '91. pp. 257–265. Springer Berlin Heidelberg, Berlin, Heidelberg (1991)
14. Chen, L., Pedersen, T.P.: New group signature schemes. In: De Santis, A. (ed.) Advances in Cryptology — EUROCRYPT'94. pp. 171–181. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
15. Chen, S., Chen, J.: Lattice-based group signatures with forward security for anonymous authentication. Heliyon **9**(4), e14917 (2023)
16. Clarisse, R., Sanders, O.: Group signature without random oracles from randomizable signatures. In: Nguyen, K., Wu, W., Lam, K.Y., Wang, H. (eds.) Provable and Practical Security. pp. 3–23. Springer International Publishing, Cham (2020)
17. Delerablée, C., Pointcheval, D.: Dynamic fully anonymous short group signatures. In: Nguyen, P.Q. (ed.) Progress in Cryptology - VIETCRYPT 2006. pp. 193–210. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
18. Derler, D., Slamanig, D.: Highly-efficient fully-anonymous dynamic group signatures. p. 551–565. ASIACCS '18, Association for Computing Machinery, New York, NY, USA (2018)
19. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Fischlin, M. (ed.) Topics in Cryptology – CT-RSA 2009. pp. 309–324. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
20. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
21. Groth, J.: Simulation-sound nizk proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) Advances in Cryptology – ASIACRYPT 2006. pp. 444–459. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
22. Guo, J., Baugh, J.P., Wang, S.: A group signature based secure and privacy-preserving vehicular communication framework. In: 2007 Mobile Networking for Vehicular Environments. pp. 103–108 (2007)

23. Kiayias, A., Yung, M.: Group signatures with efficient concurrent join. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. pp. 198–214. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
24. Kim, H., Lee, Y., Abdalla, M., Park, J.H.: Practical dynamic group signature with efficient concurrent joins and batch verifications. Journal of Information Security and Applications **63**, 103003 (2021)
25. Lee, Y.k., Han, S.w., Lee, S.j., Chung, B.h., Lee, D.G.: Anonymous authentication system using group signature. In: 2009 International Conference on Complex, Intelligent and Software Intensive Systems. pp. 1235–1239 (2009)
26. Libert, B., Mouhartem, F., Peters, T., Yung, M.: Practical "signatures with efficient protocols" from simple assumptions. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. p. 511–522. ASIA CCS '16, Association for Computing Machinery, New York, NY, USA (2016)
27. Libert, B., Yung, M.: Dynamic fully forward-secure group signatures. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. p. 70–81. ASIACCS '10, Association for Computing Machinery, New York, NY, USA (2010)
28. Lin, C., He, D., Kumar, N., Huang, X., Vijayakumar, P., Choo, K.K.R.: Homechain: A blockchain-based secure mutual authentication system for smart homes. IEEE Internet of Things Journal **7**(2), 818–829 (2020)
29. Ling, S., Nguyen, K., Wang, H., Xu, Y.: Forward-secure group signatures from lattices. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography. pp. 44–64. Springer International Publishing, Cham (2019)
30. Malina, L., Smrz, J., Hajny, J., Vrba, K.: Secure electronic voting based on group signatures. In: 2015 38th International Conference on Telecommunications and Signal Processing (TSP). pp. 6–10 (2015)
31. Perera, M., Nakamura, T., Hashimoto, M., Yokoyama, H., Cheng, C., Sakurai, K.: A survey on group signatures and ring signatures: Traceability vs. anonymity. Cryptography **6**(1) (Mar 2022), publisher Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland.
32. Pointcheval, D., Sanders, O.: Short randomizable signatures. In: Sako, K. (ed.) Topics in Cryptology - CT-RSA 2016. pp. 111–126. Springer International Publishing, Cham (2016)
33. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (Jan 1991)
34. Smart, N.P.: Cryptography Made Simple. Springer Publishing Company, Incorporated, 1st edn. (2015)
35. Song, D.X.: Practical forward secure group signature schemes. In: Proceedings of the 8th ACM Conference on Computer and Communications Security. p. 225–234. CCS '01, Association for Computing Machinery, New York, NY, USA (2001)
36. jin Song, H., Kim, T., Hwang, Y.W., Seo, D., Lee, I.Y.: A study on dynamic group signature scheme with threshold traceability for blockchain. High-Confidence Computing **4**(2), 100163 (2024)
37. Su, Q., Zhang, R., Xue, R., Sun, Y.: An efficient traceable and anonymous authentication scheme for permissioned blockchain. In: Miller, J., Stroulia, E., Lee, K., Zhang, L.J. (eds.) Web Services – ICWS 2019. pp. 110–125. Springer International Publishing, Cham (2019)
38. Sun, X., Lin, X., Ho, P.H.: Secure vehicular communications based on group signature and id-based signature scheme. In: 2007 IEEE International Conference on Communications. pp. 1539–1545 (2007)

39. Tian, H., Luo, P., Su, Y.: A group signature based digital currency system. In: Zheng, Z., Dai, H.N., Tang, M., Chen, X. (eds.) Blockchain and Trustworthy Systems. pp. 3–14. Springer Singapore, Singapore (2020)
40. Zheng, H., Zhao, Z., Zhang, X.: Access control based on group signatures in cloud service. In: 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE). vol. 2, pp. 316–320 (2012)

A Soundness Proof of NIZK in Batch Verification Mode

Lemma 4 *There exists an extractor for the NIZK in DGSS protocol in batch verification mode.*

Proof. Consider the soundness game defined in the extractability proof of Lemma 2. W.L.O.G, we assume we have two signatures from two users, u_1 and u_2, to be verified in batch. For valid signatures, the following three equations hold:

$$g_2^{s_{k_1}} = R_{1_1} \cdot T_{1_1}^c \tag{5}$$

$$g_2^{s_{k_2}} = R_{1_2} \cdot T_{1_2}^c \tag{6}$$

$$e(T_{4_1}T_{4_2}, g_2) \cdot e(h, T_{1_1}^{s_{e_1}} T_{1_2}^{s_{e_2}}) \cdot e(Y, T_{1_1}^c T_{1_2}^c) \cdot e(T_{3_1}^c T_{3_2}^c, w) = e(T_{2_1}^{s_{e_1}} T_{2_2}^{s_{e_2}}, g_2) \tag{7}$$

Equations 5 and 6 are verification equations for u_1, and u_2, respectively, while Equation 7 is the pairing batch verification equation for the two users. Assume we have two transcripts with challenges c and c', and the corresponding set of responses from u_1 and u_2, $s_{k_1}, s_{e_1}, s_{k_2}, s_{e_2}$ and $s'_{k_1}, s'_{e_1}, s'_{k_2}, s'_{e_2}$. Define the differences $\Delta c = c - c'$, $\Delta s_{k_1} = s_{k_1} - s'_{k_1}$, $\Delta s_{e_1} = s_{e_1} - s'_{e_1}$ and $\Delta s_{k_2} = s_{k_2} - s'_{k_2}$, $\Delta s_{e_2} = s_{e_2} - s'_{e_2}$. By dividing the two instances of Equation 5 for u_1, we obtain $g_2^{\Delta s_{k_1}} = T_{1_1}^{\Delta c}$ which implies $T_{1_1} = g_2^{\frac{\Delta s_{k_1}}{\Delta c}}$. Hence, we extract the value $k_1 = \frac{\Delta s_{k_1}}{\Delta c}$. For u_2, we similarly obtain $k_2 = \frac{\Delta s_{k_2}}{\Delta c}$ from Equation 6. By deviding the two instanses of Equation 7:

$$\frac{e(h, T_{1_1}^{s_{e_1}} T_{1_2}^{s_{e_2}}) \cdot e(Y, T_{1_1}^c T_{1_2}^c) \cdot e(T_{3_1}^c T_{3_2}^c, w)}{e(h, T_{1_1}^{s'_e} T_{1_2}^{s'_e}) \cdot e(Y, T_{1_1}^{c'} T_{1_2}^{c'}) \cdot e(T_{3_1}^{c'} T_{3_2}^{c'}, w)} = \frac{e(T_{2_1}^{s_e} T_{2_2}^{s_e}, g_2)}{e(T_{2_1}^{s'_e} T_{2_2}^{s'_e}, g_2)},$$

we obtain $e(h, T_{1_1}^{\frac{\Delta s_{e_1}}{\Delta c}} T_{1_2}^{\frac{\Delta s_{e_2}}{\Delta c}}) \cdot e(Y, T_{1_1} T_{1_2}) \cdot e(T_{3_1} T_{3_2}, w) = e(T_{2_1}^{\frac{\Delta s_{e_1}}{\Delta c}} T_{2_2}^{\frac{\Delta s_{e_2}}{\Delta c}}, g_2)$ which implies that if $k_1 x_1 = \frac{\Delta s_{e_1}}{\Delta c}$ and $k_2 x_2 = \frac{\Delta s_{e_2}}{\Delta c}$, the equation holds. Therefore, given the extracted k_1 and k_2, one can compute $x_1 = \frac{\Delta s_{e_1}}{\Delta s_{k_1}}$ and $x_2 = \frac{\Delta s_{e_2}}{\Delta s_{k_2}}$. Finally, since $T_{2_1} = A_1 \cdot h^{k_1}$ and $T_{2_2} = A_2 \cdot h^{k_2}$, and both k_1, k_2 and x_1, x_2 are extracted, one can extract the value of A_1 for u_1 and A_2 for u_2. Therefore, an extractor exists for the tuple (x_1, A_1, k_1) and (x_2, A_2, k_2), which implies that the zero-knowledge proof is sound.