PC-DGS: A Pool-Certified Dynamic Group Signature Scheme Based on XMSS

Qiyan Sang[†], Lin Zhou[†], and Huiyan Chen[*]

Beijing Electronic Science and Technology Institute, Beijing, China
sangqiyan123@qq.com, zl15680672501@163.com
[*]Corresponding author: Huiyan Chen (hychen2001@126.com)
[†]Qiyan Sang and Lin Zhou contributed equally to this work.

Abstract. Hash-based group signatures provide a conservative approach to post-quantum anonymous authentication, but existing constructions often face a tension between stateless signing, compact signatures, and dynamic group management. This paper proposes PC-DGS, a pool-certified dynamic group signature scheme based on XMSS-certified WOTS+ one-time credentials. The main idea is to separate offline credential certification from online message signing: the group administrator uses XMSS to certify user-generated WOTS+ public keys, while users later sign messages using lightweight one-time WOTS+ credentials. This yields a signer-side stateful but public-verifier stateless design, reducing the communication overhead inherited from SPHINCS+-style constructions. To mitigate batch-level linkability, PC-DGS introduces a shuffled index-pool mechanism for randomized XMSS leaf allocation. To prevent malicious-administrator framing, each user registers a Merkle authorization root that binds the administrator to a pre-authorized set of user-generated WOTS+ public keys. We analyze PC-DGS under standard hash-based and authenticated-encryption assumptions and establish public-verifier anonymity, traceability, non-frameability, revocation correctness, credential-reuse detectability, and signer-side forward security for used credentials. Parameterized evaluation shows that PC-DGS achieves low-kilobyte signatures, with representative profiles ranging from about 1.5 KB to 5.4 KB.

Keywords: Post-Quantum Cryptography · Dynamic Group Signature · Hash-Based Signature · XMSS

1 Introduction

Group signatures, introduced by Chaum and van Heyst [8], enable group members to sign messages anonymously on behalf of a group, while an authorized opening authority can lift anonymity when accountability is required. Their security requirements were formalized by Bellare et al. [2] under the core notions of anonymity and traceability, which were later extended to dynamic settings to accommodate group membership changes and revocation [3,6]. Most classical

schemes (e.g., [1,5,4]) rely on number-theoretic assumptions that are vulnerable to quantum algorithms. Consequently, constructing practical post-quantum group signatures has become an urgent necessity.

Post-quantum group signatures have been explored across several distinct algebraic foundations. Lattice-based and code-based constructions (e.g., [16,19,10]) offer expressive public-key models but generally require complex, heavy zero-knowledge proof systems. Symmetric-primitive and hash-based alternatives utilize conservative primitives to eliminate structural algebraic assumptions. Schemes based on MPC-in-the-Head (e.g., Sphinx-in-the-Head [9]) present general non-interactive frameworks but suffer from large proof transcripts. Conversely, tree-based and hash-based signature designs, rooted in the security of cryptographic hash functions and one-time signatures (WOTS+ [14]), offer implementation-friendly profiles with robust security bounds. Representative works in this line, including DGM [7], GMMT [22], and DGMT [12], have significantly advanced dynamic symmetric-key group authentication but often struggle with manager storage scaling or complex interaction overheads.

To address these limitations, state-of-the-art hash-based group signature systems leverage standard standardized post-quantum signature components. Standard stateful schemes like XMSS (RFC 8391 [13]) provide short signatures but mandate strict state synchronization, while stateless primitives like SPHINCS+ (FIPS 205 [20]) eliminate state management at the cost of significantly inflated signature sizes. This tension is evident in DGSP [11], the closest predecessor to our work. DGSP utilizes SPHINCS+ to establish a scalable dynamic group signature scheme; however, it inherits SPHINCS+-level communication overhead due to the inclusion of heavy FORS and hypertree components within every single signature block. Furthermore, to support comprehensive group management functionalities, DGSP renders the overall signing layer stateful. Therefore, a critical challenge remains: how to design a hash-based group signature that achieves compact signatures and stateless verification, while maintaining a manageable, well-disciplined state discipline on the signer side.

Our core contributions are summarized as follows:

- **Pool-Certified Asymmetric Configuration:** We propose PC-DGS, a post-quantum dynamic group signature scheme that introduces a pool-certified credential paradigm to achieve a "stateful signing, stateless verification" asymmetric configuration. By isolating the rigid state synchronization of the underlying XMSS tree within an asynchronous sliding-window shuffled pool, we eliminate the multi-batch linkability at the user side without disrupting tree traversal efficiency. This configuration avoids embedding heavy SPHINCS+-like hypertrees, thereby achieving more compact signatures
- **Non-frameability via Merkle Root Registration:** To defeat malicious tracking or framing by the group manager, we introduce a static Merkle authorization root mechanism registered by users upon initialization. The GA can only certify a WOTS+ public key by providing a verifiable cryptographic witness tied to this root, preventing unauthorized key binding without introducing complex zero-knowledge proofs.

- **Accountable and Scalable Management:** PC-DGS provides robust dynamic group management, enabling secure user joining, efficient credential-level revocation, and accountable identity tracing under an independent audit model. By decoupling stateless cryptographic verification from active group states, the complex management overhead is insulated from the online verification end.
- **Rigorous Analysis and Evaluation:** We analyze the security of PC-DGS under standard assumptions, confirming anonymity, traceability, non-frameability, and revocation correctness. Performance evaluations demonstrate that PC-DGS delivers KB-level compact signatures and lightweight online signing compared to SPHINCS+-based and proof-system-based alternatives.

Organization. The remainder of this paper is structured as follows. Section 2 introduces necessary preliminaries. Section 3 formalizes the detailed construction of PC-DGS. Section 4 provides the rigorous security analysis, followed by the theoretical performance evaluations in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

In this section, we formalize the foundational post-quantum and symmetric-key cryptographic primitives utilized in our construction. Throughout this paper, 1^n denotes the security parameter in unary representation, and $H : \{0,1\}^* \to \{0,1\}^{8n}$ represents a generic cryptographic hash function modeled as a random oracle.

2.1 Winternitz One-Time Signature Plus (WOTS+)

We adopt the Winternitz One-Time Signature Plus (WOTS+) scheme formalized by Hülsing [14], which serves as the core component of our dynamic group signature credential. Given the security parameter n and the Winternitz parameter $w \in \{2, 4, 16\}$, the WOTS+ scheme consists of the following three polynomial-time algorithms:

- $\mathsf{WOTS.KeyGen}(1^n) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that samples a seed to generate a private key sk and iteratively evaluates a collection of keyed hash chains to construct the corresponding public key pk.
- $\mathsf{WOTS.Sign}(\mathsf{sk}, M) \to \sigma$: An algorithm that maps an n-byte message digest M into a set of chain-specific iteration counts, computing the one-time signature σ by propagating the corresponding private key components through the hash chains.
- $\mathsf{WOTS.Verify}(\mathsf{pk}, M, \sigma) \to \{0,1\}$: A deterministic verification algorithm that reconstructs the public key candidate by completing the remaining iterations of the hash chains based on σ and M. It outputs 1 if the recomputed public key matches pk, and 0 otherwise.

WOTS+ satisfies existential unforgeability under single-query adaptive chosen-message attacks (EUF-1-CMA) without requiring collision resistance for the underlying hash function.

2.2 eXtended Merkle Signature Scheme (XMSS)

Following RFC 8391 [13], the eXtended Merkle Signature Scheme (XMSS) is a stateful hash-based signature scheme built upon a binary Merkle tree of height h_X, which can authenticate up to 2^{h_X} WOTS+ keypairs. It provides the following execution interfaces:

- $\mathsf{XMSS.KeyGen}(1^n) \rightarrow (\mathsf{pk}_{\mathrm{XMSS}}, \mathsf{sk}_{\mathrm{XMSS}})$: Generates the public key $\mathsf{pk}_{\mathrm{XMSS}}$ representing the root of the Merkle tree, and initializes the private signing state $\mathsf{sk}_{\mathrm{XMSS}}$ which tracks the non-reversible leaf index sequence.
- $\mathsf{XMSS.SignAt}(\mathsf{sk}_{\mathrm{XMSS}}, \mathsf{idx}, M) \rightarrow \mathsf{cert}$: Signs a message M using the WOTS+ secret key located at the designated leaf index $\mathsf{idx} \in [0, 2^{h_X} - 1]$. It outputs a certificate cert comprising the WOTS+ signature and the associated Merkle authentication path, while atomically advancing the internal index counter to prevent reuse.
- $\mathsf{XMSS.Verify}(\mathsf{pk}_{\mathrm{XMSS}}, M, \sigma) \rightarrow \{0, 1\}$: A stateless routine that validates the WOTS+ signature inside σ against the message M to compute a leaf candidate, and then iteratively computes the root hash via the authentication path. It returns 1 if the calculated root matches $\mathsf{pk}_{\mathrm{XMSS}}$, and 0 otherwise.

XMSS provides forward-secure existential unforgeability under adaptive chosen-message attacks (EUF-CMA) under the standard state discipline.

2.3 Authenticated Encryption with Associated Data (AEAD)

An authenticated encryption scheme with associated data (AEAD) is utilized by the group administrator to ensure both confidentiality and integrity of tracking metadata. It comprises the following symmetric primitives:

- $\mathsf{AE.Enc}_K(M; \mathsf{nonce}) \rightarrow C$: A symmetric encryption algorithm that encrypts a plaintext message M under the secret key K and a unique nonce, producing a ciphertext payload C containing a built-in authentication tag.
- $\mathsf{AE.Dec}_K(C) \rightarrow M \cup \{\perp\}$: A deterministic decryption algorithm that verifies the authenticity tag of C. If the integrity check passes, it returns the unsealed plaintext message M; otherwise, it rejects the ciphertext and outputs a verification failure token $\perp$.

The AEAD scheme is required to achieve ciphertext indistinguishability under adaptive chosen-ciphertext attacks (IND-CCA2) and ciphertext integrity (INT-CTXT).

3 Scheme Design of PC-DGS

PC-DGS is a post-quantum dynamic group signature scheme fundamentally rooted in the eXtended Merkle Signature Scheme (XMSS) certified Winternitz One-Time Signature Plus (WOTS+) credential paradigm. Architecturally, it strategically decouples group credential certification (offline/pre-authorization) from online user signature generation to avoid embedding heavy stateless signatures into every group signature, thereby reducing communication overhead.

The scheme follows a "stateful signing, stateless verification" paradigm: each WOTS+ key pair acts as a single-use credential. Honest users must securely erase or irreversibly invalidate the spent sk immediately after signing, which provides signer-side forward security for used WOTS+ credentials (subsequent signing state compromise cannot recover erased keys or generate additional signatures). Crucially, public verifiers remain completely stateless. To mitigate batch-linking risks during multiple credential acquisitions, a randomized index pool mechanism is introduced: rather than sequential allocation, the GA randomly draws k unallocated XMSS leaf indices from a shuffled pool to certify k distinct WOTS+ public keys, ensuring each leaf index is used at most once.

3.1 System Entities and Design Overview

The PC-DGS architecture is formalized via four operational entities: (1) **Group Administrator** (GA) initializing systems, maintaining epoch-level XMSS keys, managing the index pool, issuing certified credentials, tracking opening metadata, executing revocations, and opening disputed signatures; (2) **Group User** (GU) generating WOTS+ key pairs, registering a local Merkle authorization root, requesting credentials, and signing messages; (3) **Group Verifier** (GV) state-independently validating signatures via the group public key, checking XMSS/WOTS+ validity, epoch bindings, and revocation status; and (4) **Judge Authority** (JA) acting as an independent arbiter to cross-examine token consistency, XMSS/WOTS+ validity, and registered Merkle paths to guarantee non-frameability against a malicious GA under stated assumptions.

3.2 Public Parameters and System States

The system's global public parameters are defined as $\mathsf{params} = (n, w, L, K, h_X, \mathsf{enc})$, where n is the hash output byte length, w is the WOTS+ Winternitz parameter, L is the maximum pre-authorized WOTS+ keys per user, K is the initial XMSS leaf index pool size per epoch, h_X is the GA's XMSS tree height, and enc denotes the standardized fixed-length encoding rules.

The group public key is modeled as $\mathsf{gpk} = (\mathsf{params}, \mathsf{GPKList}, \mathsf{epoch}_{current})$, where $\mathsf{GPKList}[\mathsf{epoch}] = \mathsf{pk}_{\mathrm{GA},\mathsf{epoch}}$ maps each active epoch to the GA's corresponding XMSS public key. To govern the system, the GA maintains a private master state:

$$\mathsf{gmsk} = (\mathsf{sk}_{\mathrm{GA},\mathsf{epoch}_{current}}, K_T, \mathsf{Pool}, \mathsf{idx}_{max}, \mathsf{UserMap}, \mathsf{TokenMap}, \mathsf{IssuedCommit}, \mathsf{RL}) \tag{1}$$

consisting of the current epoch's private XMSS signing state $\mathsf{sk}_{\mathrm{GA},\mathsf{epoch}_{current}}$, a symmetric token encryption key K_T, the unallocated XMSS leaf index set Pool, a user identity tracking table $\mathsf{UserMap}$ mapping $uid \to \{\mathsf{cid}\}$, an opening/audit database $\mathsf{TokenMap}$ indexed by cid, a deduplication commitment set $\mathsf{IssuedCommit}$, and the active revocation list RL.

In addition, an immutable global registration ledger records $\mathsf{RegList}[uid] = \mathsf{Root}_u$, binding each user to a fixed Merkle authorization root Root_u under a non-substitutability property. The user's local state is modeled as a stateful dynamic collection $\mathsf{gsk}_u = \{(\mathsf{pk}_i, \mathsf{sk}_i, r_i, \mathsf{path}_i, \mathsf{status}_i, \mathsf{cred}_i)\}_{i=1}^{L}$, where $\mathsf{status}_i \in \{\mathsf{fresh}, \mathsf{certified}, \mathsf{used}\}$. For fresh entries, the credential slot $\mathsf{cred}_i = \perp$, which is populated after Join or Update. Every signature generation forces the secure erasure or irreversible invalidation of the spent sk_i, transitioning its status to used.

3.3 Credential Structure, Signature Format, and Derived Values

An active user credential and its corresponding public group signature Σ are formalized as:

$$\mathsf{cred} = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, \mathsf{sk}, c, \mathsf{tok}, \mathsf{cert}) \tag{2}$$

$$\Sigma = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}, \sigma) \tag{3}$$

where $(\mathsf{pk}, \mathsf{sk})$ is the WOTS+ key pair, c is the authorization commitment, tok is the authenticated encryption (AE) token for tracing, cert is the GA-issued XMSS certificate, and σ is the WOTS+ one-time signature. During protocol execution, the interactive cryptographic binding chain is established via the derived values specified in Table 1.

Table 1. Cryptographic Derived Values and Chained Bindings

Derived Component	Mathematical Formulation / Cryptographic Definition
Identity Tag	$uidTag = H_{id}(uid)$
Authorization Commitment	$c = H(\mathsf{pk} \parallel r)$ where $r \leftarrow \{0,1\}^{8n}$
Token Plaintext & Ciphertext	$\mathsf{TokenPlain} = uidTag \parallel \mathsf{epoch} \parallel \mathsf{idx} \parallel c$; $\mathsf{tok} = \mathsf{AE.Enc}_{K_T}(\mathsf{TokenPlain}; \mathsf{nonce})$
Certificate Message	$\mathsf{CertMsg} = \mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok}$
Credential Identifier	$\mathsf{cid} = H(\mathsf{CertMsg}) = H(\mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok})$
WOTS+ Signing Message	$\mathsf{SigMsg} = H(m \parallel ctx \parallel \mathsf{cid})$ for message m and context ctx

3.4 Algorithm GSetup

The system initialization algorithm $\mathsf{GSetup}(n, w, L, K, h_X)$ is executed by the GA to boot the system parameters:

1. Set $\mathsf{params} = (n, w, L, K, h_X, \mathsf{enc})$ and $\mathsf{epoch}_{current} = 0$.

2. Generate initial XMSS keypair: $(\mathsf{pk}_{\mathrm{GA},0}, \mathsf{sk}_{\mathrm{GA},0}) \leftarrow \mathsf{XMSS.KeyGen}(1^n)$, and assign $\mathsf{GPKList}[0] = \mathsf{pk}_{\mathrm{GA},0}$.
3. Sample a master symmetric encryption key: $K_T \leftarrow \{0,1\}^{8n}$.
4. Initialize the precomputed credential pool Pool of capacity K by sequentially generating the XMSS authentication paths auth_i for the initial K leaf nodes ($K \leq 2^{h_X}$), setting $\mathsf{Pool} \leftarrow \{(i, \mathsf{auth}_i)\}_{i=0}^{K-1}$, and initialize the maximum index counter $\mathsf{idx}_{max} \leftarrow K-1$.
5. Set $\mathsf{UserMap}, \mathsf{TokenMap}, \mathsf{IssuedCommit}, \mathsf{RL} \leftarrow \emptyset$. Output $\mathsf{gpk} = (\mathsf{params}, \mathsf{GPKList}, 0)$ and private state $\mathsf{gmsk} = (\mathsf{sk}_{\mathrm{GA},0}, K_T, \mathsf{Pool}, \mathsf{idx}_{max}, \mathsf{UserMap}, \mathsf{TokenMap}, \mathsf{IssuedCommit}, \mathsf{RL})$.

3.5 Algorithm URegister

The user registration algorithm $\mathsf{URegister}(uid)$ is executed locally by a group user uid to build a pre-authorized credential pool:

1. Generate L WOTS+ keypairs: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{WOTS.KeyGen}(1^n)$ for $i \in \{1, \ldots, L\}$.
2. Sample random salts $r_i \leftarrow \{0,1\}^{8n}$ and compute commitments: $c_i = H(\mathsf{pk}_i \parallel r_i)$.
3. Construct a local Merkle authorization tree root: $\mathsf{Root}_u = \mathsf{MerkleRoot}(c_1, c_2, \ldots, c_L)$.
4. Publish the non-substitutable binding ledger entry: $\mathsf{RegList}[uid] = \mathsf{Root}_u$.
5. Retain local state $\mathsf{gsk}_u = \{(\mathsf{pk}_i, \mathsf{sk}_i, r_i, \mathsf{path}_i, \mathsf{status}_i = \mathsf{fresh}, \mathsf{cred}_i = \perp)\}_{i=1}^{L}$, where path_i is the authentication path from c_i to Root_u.

3.6 Algorithm Join

The interactive credential activation protocol $\mathsf{Join}(\mathsf{gsk}_u, \mathsf{gmsk})$ executes via an authenticated channel to allocate precomputed path tuples from the dynamically replenished credential pool:

1. User transmits request packet $(uid, \{(\mathsf{pk}_j, r_j, \mathsf{path}_j)\}_{j=1}^{k})$ for k distinct slots marked as fresh.
2. GA retrieves $\mathsf{Root}_u = \mathsf{RegList}[uid]$ (aborts if missing) and verifies $\mathsf{MerkleVerify}(\mathsf{Root}_u, H(\mathsf{pk}_j \parallel r_j), \mathsf{path}_j) = 1$ for all $j \in \{1, \ldots, k\}$.
3. GA enforces deduplication by asserting $c_i \neq c_j$ $(\forall i \neq j)$ and cross-checking $c_j \notin \mathsf{IssuedCommit}$.
4. GA randomly samples k distinct credential tuples from the precomputed pool without replacement to eliminate batch linkability:

$$\{(\mathsf{idx}_1, \mathsf{auth}_1), \ldots, (\mathsf{idx}_k, \mathsf{auth}_k)\} \leftarrow \mathsf{Sample}(\mathsf{Pool}, k)$$

   and temporarily removes them from the active set: $\mathsf{Pool} \leftarrow \mathsf{Pool} \backslash \{(\mathsf{idx}_j, \mathsf{auth}_j)\}_{j=1}^{k}$.
5. GA sequentially processes each slot $j \in \{1, \ldots, k\}$. The dynamic authority certificate cert_j is instantly assembled by evaluating a WOTS+ signature $\sigma_{\mathrm{WOTS},j}$ over $\mathsf{CertMsg}_j$ under the idx_j-th leaf private key and appending the

sampled path auth_j:

$$uidTag = H_{id}(uid),\ \mathsf{TokenPlain}_j = uidTag \parallel \mathsf{epoch}_{current} \parallel \mathsf{idx}_j \parallel c_j$$
$$\mathsf{tok}_j = \mathsf{AE.Enc}_{K_T}(\mathsf{TokenPlain}_j; \mathsf{nonce}_j)$$
$$\mathsf{CertMsg}_j = \mathsf{epoch}_{current} \parallel \mathsf{idx}_j \parallel \mathsf{pk}_j \parallel c_j \parallel \mathsf{tok}_j$$
$$\mathsf{cid}_j = H(\mathsf{CertMsg}_j),\ \sigma_{\mathrm{WOTS},j} \leftarrow \mathsf{WOTS.Sign}(\mathsf{sk}_{\mathrm{GA},\mathsf{epoch}_{current}}^{\mathsf{idx}_j}, \mathsf{CertMsg}_j)$$
$$\mathsf{cert}_j = (\sigma_{\mathrm{WOTS},j} \parallel \mathsf{auth}_j)$$

6. GA updates the master logs: $\mathsf{IssuedCommit} \leftarrow \mathsf{IssuedCommit} \cup \{c_j\}$, $\mathsf{UserMap}[uid] \leftarrow \mathsf{UserMap}[uid] \cup \{\mathsf{cid}_j\}$, and records the complete structural tuples in $\mathsf{TokenMap}[\mathsf{cid}_j]$.
7. To maintain a constant pool capacity K, GA sequentially precomputes k fresh trailing XMSS authentication paths starting from the historical high-watermark $\mathsf{idx}_{max} + 1$. Specifically, GA updates the pool and the top index tracker as follows:

$$\mathsf{Pool} \leftarrow \mathsf{Pool} \cup \{(i, \mathsf{auth}_i)\}_{i=\mathsf{idx}_{max}+1}^{\mathsf{idx}_{max}+k}$$
$$\mathsf{idx}_{max} \leftarrow \mathsf{idx}_{max} + k$$

8. GA returns $\{(\mathsf{epoch}_{current}, \mathsf{idx}_j, c_j, \mathsf{tok}_j, \mathsf{cert}_j)\}_{j=1}^{k}$. User mutates local entry state to $\mathsf{status}_j = \mathsf{certified}$ and materializes $\mathsf{cred}_j = (\mathsf{epoch}_{current}, \mathsf{idx}_j, \mathsf{pk}_j, \mathsf{sk}_j, c_j, \mathsf{tok}_j, \mathsf{cert}_j)$.

3.7 Algorithm Update

The replenishment algorithm $\mathsf{Update}(\mathsf{gsk}_u, \mathsf{gmsk})$ executes identically to Join to activate additional credentials under the permanent cryptographic invariant that the immutable registration root $\mathsf{RegList}[uid] = \mathsf{Root}_u$ is never altered.

3.8 Algorithm GSign

The stateful group signature generation algorithm $\mathsf{GSign}(\mathsf{gsk}_u, m, ctx) \to \Sigma$ proceeds as follows:

1. Parse an active credential $\mathsf{cred} = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, \mathsf{sk}, c, \mathsf{tok}, \mathsf{cert})$ from gsk_u.
2. Compute credential identifier $\mathsf{cid} = H(\mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok})$ and message digest $\mathsf{SigMsg} = H(m \parallel ctx \parallel \mathsf{cid})$.
3. Invoke WOTS+ one-time signature: $\sigma \leftarrow \mathsf{WOTS.Sign}(\mathsf{sk}, \mathsf{SigMsg})$.
4. Output $\Sigma = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}, \sigma)$, instantly wipe sk from local memory, and update its status to used.

3.9 Algorithm GVerify

The stateless verification pipeline $\mathsf{GVerify}(\mathsf{gpk}, m, ctx, \Sigma) \to \{0, 1\}$ checks credential validity via the following rules:

1. Abort if $\mathsf{epoch} \notin \mathsf{GPKList}$. Otherwise, extract the public key: $\mathsf{pk}_{\mathrm{GA},\mathsf{epoch}} = \mathsf{GPKList}[\mathsf{epoch}]$.

2. Reconstruct $\mathsf{cid} = H(\mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok})$. If $\mathsf{cid} \in \mathsf{RL}$, return 0 (rejected).
3. Assert authority certification via XMSS: $\mathsf{XMSS.Verify}(\mathsf{pk}_{\mathrm{GA},\mathsf{epoch}}, \mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok}, \mathsf{cert}) \stackrel{?}{=} 1$.
4. Assert primitive signature validity via WOTS+: $\mathsf{WOTS.Verify}(\mathsf{pk}, H(m \parallel ctx \parallel \mathsf{cid}), \sigma) \stackrel{?}{=} 1$. Return 1 if both hold.

3.10 Algorithm Open

The signature tracing algorithm $\mathsf{Open}(\mathsf{gmsk}, \Sigma) \to \pi_{open}$ unseals identity information via the following operations:

1. Compute $\mathsf{cid} = H(\mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok})$ and decrypt the encrypted tracking container: $\mathsf{AE.Dec}_{K_T}(\mathsf{tok}) \to uidTag' \parallel \mathsf{epoch}' \parallel \mathsf{idx}' \parallel c'$.
2. Assert parameter integrity: $\mathsf{epoch}' = \mathsf{epoch}$, $\mathsf{idx}' = \mathsf{idx}$, and $c' = c$.
3. Fetch the matching audit log from $\mathsf{TokenMap}[\mathsf{cid}]$ to retrieve user identity uid, salt r, and path path, confirming $uidTag' = H_{id}(uid)$.
4. Output the cryptographic opening proof: $\pi_{open} = (uid, uidTag, \mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cid}, r, \mathsf{path})$.

3.11 Algorithm Judge

The auditing routine $\mathsf{Judge}(m, ctx, \Sigma, \pi_{open}) \to \{0, 1\}$ validates tracing results under the Independent Audit Model:

1. Check field structural alignment between π_{open} and Σ, and assert $uidTag = H_{id}(uid)$.
2. Validate token integrity via the independent interface: $\mathsf{AE.Dec}_{K_T}(\mathsf{tok}) \stackrel{?}{=} uidTag \parallel \mathsf{epoch} \parallel \mathsf{idx} \parallel c$, preventing the GA from fabricating the decryption result.
3. Recompute references cid, CertMsg, and evaluate dual mathematical verifications: $\mathsf{XMSS.Verify}(\mathsf{pk}_{\mathrm{GA},\mathsf{epoch}}, \mathsf{CertMsg}, \mathsf{cert}) = 1$ and $\mathsf{WOTS.Verify}(\mathsf{pk}, H(m \parallel ctx \parallel \mathsf{cid}), \sigma) = 1$.
4. Retrieve $\mathsf{Root}_u = \mathsf{RegList}[uid]$, and verify commitment paths: $c = H(\mathsf{pk} \parallel r)$ and $\mathsf{MerkleVerify}(\mathsf{Root}_u, c, \mathsf{path}) = 1$. Return 1 upon complete success.

3.12 Algorithm Revoke

The fine-grained credential revocation algorithm $\mathsf{Revoke}(\mathsf{gmsk}, uid)$ executes as follows:

1. Query the administrative multi-map ledger: $\mathsf{UserMap}[uid] \to C_{uid} = \{\mathsf{cid}_1, \ldots, \mathsf{cid}_t\}$.
2. Append the target user's active credential space to the blacklist: $\mathsf{RL} \leftarrow \mathsf{RL} \cup C_{uid}$.

It is worth noting that besides full identity-level expulsion, our construction natively supports fine-grained revocation at the granularity of a specific cid (i.e., an individual leaf node). Although standard policies typically mandate total user erasure, the architectural flexibility of PC-DGS enables the GA to selectively append only a single compromised cid to RL if necessary, strictly preserving the validity of the user's remaining fresh or certified credentials.

3.13 Algorithm EpochRefresh

The epoch evolution algorithm $\mathsf{EpochRefresh}(\mathsf{gmsk})$ rotates keys upon cycle expiration:

1. Update $\mathsf{epoch}_{current} \leftarrow \mathsf{epoch}_{current} + 1$.
2. Generate a fresh master signing keypair: $(\mathsf{pk}_{\mathrm{GA},\mathsf{epoch}_{current}}, \mathsf{sk}_{\mathrm{GA},\mathsf{epoch}_{current}}) \leftarrow \mathsf{XMSS.KeyGen}(1^n)$, and register the new instance into $\mathsf{GPKList}$.
3. Allocate a fully reset precomputed credential pool Pool of capacity K by sequentially generating the XMSS authentication paths auth_i under the fresh private key $\mathsf{sk}_{\mathrm{GA},\mathsf{epoch}_{current}}$, setting $\mathsf{Pool} \leftarrow \{(i, \mathsf{auth}_i)\}_{i=0}^{K-1}$, and reset the maximum leaf index counter $\mathsf{idx}_{max} \leftarrow K - 1$.

3.14 Correctness Analysis

The completeness of PC-DGS holds under the assumption that all parties honestly follow the protocol specifications.

During registration ($\mathsf{URegister}$), the user establishes a Merkle root Root_u derived from commitments $c_i = H(\mathsf{pk}_i \parallel r_i)$. In subsequent Join protocols, the payload tuple $(\mathsf{pk}_j, r_j, \mathsf{path}_j)$ evaluates to $\mathsf{MerkleVerify}(\mathsf{Root}_u, c_j, \mathsf{path}_j) = 1$ by construction.

Because the GA issues certificates exclusively for pre-authorized public keys, the signature cert spanning $\mathsf{CertMsg} = \mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok}$ generated via the active XMSS signing key evaluates to $\mathsf{XMSS.Verify}(\mathsf{pk}_{\mathrm{GA},\mathsf{epoch}}, \mathsf{CertMsg}, \mathsf{cert}) = 1$ under the standard completeness of XMSS.

Furthermore, during online signature generation (GSign), the signer evaluates a standard WOTS+ primitive signature over $\mathsf{SigMsg} = H(m \parallel ctx \parallel \mathsf{cid})$ using the exact secret key sk matched to the certified public key pk. Thus, by the correctness of WOTS+, the stateless evaluation routine yields $\mathsf{WOTS.Verify}(\mathsf{pk}, \mathsf{SigMsg}, \sigma) = 1$.

Finally, during judicial dispute arbitration (Open), the non-malleability of the authenticated encryption scheme ensures that the GA reconstructs the identical token fields from tok and furnishes a valid opening proof π_{open} via database inspection. Consequently, when the JA re-evaluates the token decryption structure, the authority certificate, the primitive signature, and the immutable registration path, all cascaded algebraic constraints evaluate to true, completing the proof of correctness.

4 Security Analysis

This section presents a concise game-based security analysis of PC-DGS. Since the construction is built from standard XMSS, WOTS+, and authenticated encryption components, our proofs are reduction-oriented rather than tight. Polynomial query factors are omitted for readability. The goal is to establish the core group-signature properties required by our construction: public-verifier anonymity, traceability against malicious users under an honest opening authority, non-frameability against a malicious group administrator, and revocation correctness.

4.1 Security Model and Assumptions

Let n denote the hash-output length in bytes. The hash functions H and H_{id} are assumed to be preimage resistant and collision resistant, and are modeled as random oracles when they are used for challenge binding. WOTS+ is assumed to be existentially unforgeable for one-time use. XMSS is assumed to be existentially unforgeable under correct state management and non-reuse of leaf indices. The authenticated encryption scheme AE is assumed to satisfy IND-CCA confidentiality and INT-CTXT integrity.

The public registration ledger $\mathsf{RegList}$ is assumed to be non-substitutable: once a user *uid* registers Root_u, the group administrator cannot replace, fork, or equivocate this value. The Join and Update protocols are executed over authenticated and confidential channels, since the tuple $(uid, \mathsf{pk}, r, \mathsf{path})$ explicitly links a user to a future public credential. The randomized index-pool allocation is assumed to be identity-independent, meaning that the distribution of allocated indices is independent of the requesting user's identity except through public system parameters.

PC-DGS distinguishes public verification from authorized opening. Public verifiers do not hold the token key K_T, and therefore cannot decrypt tok. The group administrator and the judge authority are authorized opening or auditing entities. In particular, the judge authority may hold K_T, or an equivalent token-audit capability, for verifying disputed openings. Therefore, anonymity is claimed only against public verifiers and external adversaries without K_T.

PC-DGS is signer-side stateful. Each certified WOTS+ credential is intended for one-time use. Honest users sign only with credentials marked as $\mathsf{certified}$, securely erase or irreversibly invalidate the corresponding WOTS+ secret key after signing, and update the local status to used. Public verifiers do not maintain a global spent list. If a malicious user deliberately backs up and reuses its own WOTS+ secret key, the reuse is publicly detectable rather than cryptographically prevented at verification time.

4.2 Public-Verifier Anonymity

Experiment $\mathsf{Exp}^{\mathsf{anon}}_{\mathcal{A}}(1^n)$. The challenger runs GSetup and gives gpk to $\mathcal{A}$. The adversary may adaptively query registration, joining, signing, verification, and revocation oracles, but it is not given K_T and does not obtain the confidential join/update transcripts of the challenge users. Eventually, $\mathcal{A}$ outputs two registered and non-revoked users (u_0, u_1), a message m, and a context ctx. The challenger samples $b \leftarrow \{0,1\}$, generates a valid challenge signature Σ^* using a fresh certified credential of u_b, and returns Σ^* to $\mathcal{A}$. The adversary outputs b'. Its advantage is

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{A}} = \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

Theorem 1 (Public-verifier anonymity). *If* AE *is IND-CCA secure, the join/update transcripts are confidential, and index allocation is identity-independent,*

then for every PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{AE}}(\mathcal{B}) + \mathsf{negl}(n).$$

Proof. A challenge signature has the form

$$\Sigma^* = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}, \sigma).$$

The only identity-dependent value in the public signature is $uidTag = H_{id}(uid)$, which appears only inside

$$\mathsf{tok} = \mathsf{AE.Enc}_{K_T}(uidTag \parallel \mathsf{epoch} \parallel \mathsf{idx} \parallel c; \mathsf{nonce}).$$

Since the adversary does not possess K_T, distinguishing whether tok encrypts the identity tag of u_0 or u_1 yields an IND-CCA distinguisher against AE. The remaining public fields pk, c, cert, and σ do not explicitly contain the user identity. The field idx is sampled from an identity-independent randomized pool, and therefore does not encode the requester identity. Finally, the mapping between users and credentials is hidden by the confidential join/update channel. Hence the adversary's advantage is negligible under the stated assumptions.

4.3 Traceability Against Malicious Users

Experiment $\mathsf{Exp}^{\mathsf{trace}}_{\mathcal{A}}(1^n)$. The challenger runs GSetup and gives gpk to $\mathcal{A}$. The adversary may corrupt users and may query joining, signing, opening, and revocation oracles. The opening authority is assumed honest and maintains consistent issuance records. The adversary wins if it outputs (m, ctx, Σ) such that

$$\mathsf{GVerify}(\mathsf{gpk}, m, ctx, \Sigma) = 1,$$

but the honest Open algorithm fails to output a registered user or the resulting opening proof is rejected by Judge. Let the winning probability be $\mathsf{Adv}^{\mathsf{trace}}_{\mathcal{A}}$.

Theorem 2 (Traceability against malicious users). *Under an honest opening authority, for every PPT adversary $\mathcal{A}$, there exist PPT adversaries $(\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ such that*

$$\mathsf{Adv}^{\mathsf{trace}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{euf}}_{\mathsf{XMSS}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{AE}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{coll}}_{H}(\mathcal{B}_3) + \mathsf{negl}(n).$$

Proof. Let

$$\Sigma = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}, \sigma)$$

be an accepted signature. Acceptance by $\mathsf{GVerify}$ implies that cert is a valid XMSS certificate on

$$\mathsf{CertMsg} = \mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok}.$$

If this certified credential was never issued by the honest opening authority, then the adversary has produced a forgery against XMSS. Otherwise, the credential

has a corresponding issuance record indexed by $\mathsf{cid} = H(\mathsf{CertMsg})$, except with negligible probability of breaking the collision resistance of H (which acts as a unique database key). If tok is modified so that it decrypts to a different valid identity-binding tuple, the adversary violates the INT-CTXT security of AE. Furthermore, an adversary cannot find another distinct message $\mathsf{CertMsg}' \neq \mathsf{CertMsg}$ such that $H(\mathsf{CertMsg}') = H(\mathsf{CertMsg}) = \mathsf{cid}$, which would otherwise cause the judge authority to evaluate cryptographic constraints against an incorrect user identity during arbitration. Therefore, any accepted non-revoked signature corresponds to a well-defined issuance record in $\mathsf{TokenMap}$. The honest opening authority can recover the registered signer identity and produce a judge-accepted opening proof.

4.4 Non-Frameability

Experiment $\mathsf{Exp}^{\mathsf{nf}}_{\mathcal{A}}(1^n)$. The adversary controls the group administrator and may issue credentials, choose tokens, produce signatures, and generate opening proofs. It may also corrupt any subset of group users except the target honest user u. For corrupted users, the adversary obtains their local signing states, certified credentials, salts, Merkle witnesses, and any other user-side secret material. Honest users generate WOTS+ secret keys locally and register non-substitutable Merkle authorization roots. The adversary wins if it outputs

$$(m, ctx, \Sigma, \pi_{open}, u)$$

such that Judge accepts the opening of Σ to the target honest user u, while the WOTS+ public key in Σ was not authorized under u's registered root, or the signature was not generated using a credential authorized by u. Let the winning probability be $\mathsf{Adv}^{\mathsf{nf}}_{\mathcal{A}}$.

Theorem 3 (Non-frameability). *For every PPT adversary $\mathcal{A}$, there exist PPT adversaries $(\mathcal{B}_1, \mathcal{B}_2)$ such that*

$$\mathsf{Adv}^{\mathsf{nf}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{bind}}_{\mathsf{Merkle}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{euf\text{-}1}}_{\mathsf{WOTS+}}(\mathcal{B}_2) + \mathsf{negl}(n).$$

Proof. For Judge to accept an opening to u, the proof must satisfy

$$\mathsf{RegList}[u] = \mathsf{Root}_u, \quad c = H(\mathsf{pk} \parallel r), \quad \mathsf{MerkleVerify}(\mathsf{Root}_u, c, \mathsf{path}) = 1.$$

Since $\mathsf{RegList}$ is non-substitutable, the malicious administrator cannot replace the target honest user's registered root after registration.

The additional corruption of other group users does not help in framing u. Even if corrupted users reveal their WOTS+ secret keys, certified credentials, salts, and Merkle witnesses to the malicious administrator, these values are bound to their own registered authorization roots and do not provide a valid Merkle witness under Root_u.

If pk was not authorized by u, then producing r and path such that $c = H(\mathsf{pk} \parallel r)$ is accepted under Root_u breaks the binding of the Merkle authorization tree. If pk was authorized by u but the honest user did not generate the

signature, then producing a valid WOTS+ signature under pk breaks WOTS+ one-time unforgeability, since the corresponding WOTS+ secret key is generated and retained locally by the honest user and is never disclosed to the administrator or to corrupted users. Thus, successful framing is negligible.

4.5 Revocation and Credential-State Properties

Proposition 1 (Revocation correctness). *Under a fixed current revocation state, if a credential identifier* cid *is included in* RL*, then any signature using that credential is rejected by* $\mathsf{GVerify}$*.*

Proof. $\mathsf{GVerify}$ deterministically recomputes

$$\mathsf{cid} = H(\mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok})$$

from the public signature fields and rejects immediately if $\mathsf{cid} \in \mathsf{RL}$. Therefore, no signature using a revoked credential can be accepted under the current revocation list. Historical verification can be handled by timestamped or epoch-specific snapshots of RL.

Credential-reuse detectability. PC-DGS does not claim verifier-side prevention of credential reuse. Instead, if the same certified WOTS+ credential is reused, both signatures contain the same public tuple $(\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert})$ and therefore the same cid. Reuse is thus publicly detectable without requiring a verifier-side signing list.

Signer-side forward security. After an honest signing operation, the used WOTS+ secret key is securely erased or irreversibly invalidated, and the local status is updated to used. A later compromise of the remaining signer-side state does not reveal erased WOTS+ secret keys or enable additional signatures under past used credentials. This property does not protect malicious users who intentionally back up their secret keys before erasure.

5 Performance Analysis

This section provides a theoretical evaluation of PC-DGS. Let S_{sig} and S_{gpk} denote the signature and group public key sizes, and let T_{sign} and T_{verify} be the online signing and verification times. We use n for the hash output length, ℓ_w for the WOTS+ chain count, and h_X for the XMSS tree height. Let b_e and b_i be the byte lengths of epoch and idx, while ν and τ represent the AEAD nonce and tag lengths.

A public signature in PC-DGS is $\Sigma = (\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}, \sigma)$. Under compact WOTS+ public-key representation, the total signature size is computed as:

$$S_{\mathsf{sig}}^{\mathsf{PC\text{-}DGS}} = (2\ell_w + h_X + 4)n + 2b_e + 2b_i + \nu + \tau \approx \Theta((2\ell_w + h_X)n).$$

If full WOTS+ public keys are transmitted, S_{sig} increases by $((\ell_w - 1)n)$. Table 2 details the asymptotic and concrete complexity profiles of the system.

Table 2. Theoretical cost and complexity profile of PC-DGS.

Metric / Component	Cost / Complexity Formulation
Signature size (S_{sig})	$(2\ell_w + h_X + 4)n + 2b_e + 2b_i + \nu + \tau$ (Dominant: $\Theta((2\ell_w + h_X)n)$)
Online signing time (T_{sign})	$T_{\mathsf{WOTS.Sign}} = O(\ell_w(w-1))$
Public verification time (T_{verify})	$T_{\mathsf{XMSS.Verify}} + T_{\mathsf{WOTS.Verify}} = O(2\ell_w(w-1) + h_X)$
Opening / Judging time	$T_{\mathsf{open}} = T_{\mathsf{AE.Dec}} + T_{\mathsf{lookup}}$, $T_{\mathsf{judge}} = T_{\mathsf{verify}} + T_{\mathsf{AE.Dec}} + O(\log L)$
Join/Update cost per credential	$T_{\mathsf{XMSS.SignAt}} + O(\log L) = O(\ell_w(w-1) + h_X + \log L)$
User-side storage state	$O\big(L(\ell_w n + \log L \cdot n) + C((\ell_w + h_X)n + \lvert\mathsf{tok}\rvert + b_e + b_i + n)\big)$
GA records (Q credentials)	$O(Q)$ entries in UserMap, TokenMap, and IssuedCommit
GA Index pool / Revocation list	$O(K(h_X n + b_i))$ (precomputed paths), $O(Rn)$ (for R revoked credentials)

5.1 Parameter Instantiation and Evaluation

We instantiate PC-DGS using $w = 16$, $b_e = 4$, $b_i = 4$, $\nu = 12$, and $\tau = 16$. The group public key for a single active epoch is $S_{\mathsf{gpk}} \approx 2n + b_e$. For $w = 16$, the chain configurations are $\ell_w = 35, 51, 67$ for security levels $n = 16, 24, 32$, respectively. Table 3 presents concrete sizes across different NIST security targets.

Table 3. Representative parameter instantiations of PC-DGS.

Profile	Target Security	n	w	ℓ_w	h_X	Capacity/Epoch	Sug. L	Sug. K	S_{gpk}	Comp. pk	S_{sig}
PC-DGS-128-S	NIST-I / 128-bit	16	16	35	20	2^{20}	2^{10}	2^{12}	36 B	16 B	1548 B
PC-DGS-128-L	NIST-I / 128-bit	16	16	35	24	2^{24}	2^{12}	2^{14}	36 B	16 B	1612 B
PC-DGS-192	NIST-III / 192-bit	24	16	51	24	2^{24}	2^{12}	2^{14}	52 B	24 B	3164 B
PC-DGS-256	NIST-V / 256-bit	32	16	67	24	2^{24}	2^{12}	2^{14}	68 B	32 B	5228 B
PC-DGS-256-L	NIST-V / 256-bit	32	16	67	32	2^{32}	2^{16}	2^{16}	68 B	32 B	5484 B

The data shows that compact PC-DGS signatures remain in the low-kilobyte range (1.5–5.4 KB) across all profiles. Increasing tree height h_X expands credential capacity exponentially while adding only n bytes per tree level, avoiding the need for a SPHINCS+-style hypertree structure. Online signing requires only one user-side WOTS+ operation, shifting the XMSS state synchronization overhead entirely to the asynchronous offline credential-issuance phase.

5.2 Comparison with Hash-Based and Symmetric Alternatives

Table 4 compares PC-DGS with state-of-the-art dynamic symmetric-key and hash-based group signatures. For SPHINCS+-based DGSP, the dominant signature component is bounded by $S_{\mathsf{SPHINCS+}} = n(1 + k_F(a+1) + h_S + d\ell_S)$ plus management metadata S_{meta}, where its size and signing cost are tied to the heavy FORS layer and multi-layer hypertree traversal.

By extracting the certification layer to the signer-side state via XMSS-certified WOTS+ paths, PC-DGS reduces signature size significantly compared to DGSP, presenting a strategic tradeoff: DGSP provides stateless signing at the expense of large signatures, whereas PC-DGS achieves more compact signatures and lightweight online signing under a disciplined signer-side state model.

Table 4. Comparison with dynamic hash-based and symmetric-primitive group signatures.

Scheme	Type	Sig. Size	Dominant Size Term	Signing (T_{sign})	Verification (T_{verify})
PC-DGS	Hash-based, dynamic	1.5–5.4 KB	$\Theta((2\ell_w + h_X)n)$	$T_{\mathsf{WOTS.Sign}}$	$T_{\mathsf{XMSS.Verify}} + T_{\mathsf{WOTS.Verify}}$
DGSP [11]	SPHINCS+-based, dynamic	8.27–31.27 KB	$S_{\mathsf{SPHINCS+}} + S_{\mathsf{meta}}$	$T_{\mathsf{FORS.Sign}} + dT_{\mathsf{WOTS.Sign}} + T_{\mathsf{HT}}$	$T_{\mathsf{FORS.Verify}} + dT_{\mathsf{WOTS.Verify}} + T_{\mathsf{HT}}$
DGMT [12]	Sym./hash-based	5.34–5.75 KB	S_{DGMT}	$T_{\mathsf{DGMT.Sign}}$	$T_{\mathsf{DGMT.Verify}}$
SITH [9]	MPC-in-the-Head	571–3483 KB	$O(\rho(S_{\mathsf{commit}} + S_{\mathsf{resp}}))$	$O(\rho T_{\mathsf{MPC}})$	$O(\rho T_{\mathsf{check}})$

Furthermore, PC-DGS approaches the optimal signature range of dynamic symmetric-key designs (DGMT) while offering a stronger non-frameability architecture via user-registered Merkle roots. It also achieves orders-of-magnitude communication reductions compared to proof-transcript-based MPC-in-the-Head constructions (SITH). Lattice-based alternatives [15,21,18,17] commonly rely on different dynamicity assumptions or scale asymptotically as $O(\lambda \log N)$ under heavy proof systems, positioning PC-DGS as a highly practical and lightweight candidate in the post-quantum dynamic group signature design space.

6 Conclusion

This paper presented PC-DGS, a pool-certified dynamic group signature scheme built from XMSS-certified WOTS+ one-time credentials. By decoupling offline credential certification from online user signing, PC-DGS uses XMSS only at the group-authority side to certify user-generated WOTS+ public keys, while online signatures require only one WOTS+ signing operation. This design realizes a signer-side stateful and public-verifier stateless architecture, avoiding verifier-side spent lists while preserving public verification, authorized opening, dynamic membership, and credential-level revocation.

The scheme further introduces two mechanisms tailored to the hash-based dynamic setting. First, the shuffled index-pool mechanism randomizes XMSS leaf allocation and mitigates deterministic batch-level linkability during credential issuance. Second, the Merkle authorization-root mechanism binds each user to a pre-authorized set of locally generated WOTS+ public keys, preventing a malicious group administrator from certifying unauthorized keys and framing an honest user. Under standard assumptions on WOTS+, XMSS, hash functions, and authenticated encryption, PC-DGS satisfies public-verifier anonymity, traceability against malicious users under an honest opening authority, non-frameability against a malicious administrator, revocation correctness, credential-reuse detectability, and signer-side forward security for used credentials.

The performance analysis shows that PC-DGS has a compact communication profile dominated by two WOTS+ signatures and one XMSS authentication path. Representative parameter choices yield signatures in the low-kilobyte range, making PC-DGS a practical hash-based design for post-quantum dynamic group authentication.

References

1. 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)
2. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques. p. 614–629. EUROCRYPT'03, Springer-Verlag, Berlin, Heidelberg (2003)
3. 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)
4. 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)
5. 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). https://doi.org/10.1145/1030083.1030106, https://doi.org/10.1145/1030083.1030106
6. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J.: Foundations of fully dynamic group signatures. J. Cryptol. **33**(4), 1822–1870 (Oct 2020). https://doi.org/10.1007/s00145-020-09357-w, https://doi.org/10.1007/s00145-020-09357-w
7. Buser, M., Liu, J.K., Steinfeld, R., Sakzad, A., Sun, S.F.: Dgm: A dynamic and revocable group merkle signature. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) Computer Security – ESORICS 2019. pp. 194–214. Springer International Publishing, Cham (2019)
8. 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)
9. Chen, L., Dong, C., Newton, C.J.P., Wang, Y.: Sphinx-in-the-head: Group signatures from symmetric primitives. ACM Trans. Priv. Secur. **27**(1) (Feb 2024). https://doi.org/10.1145/3638763, https://doi.org/10.1145/3638763
10. Ezerman, M.F., Lee, H.T., Ling, S., Nguyen, K., Wang, H.: A provably secure group signature scheme from code-based assumptions. Cryptology ePrint Archive, Paper 2015/479 (2015), https://eprint.iacr.org/2015/479
11. Fadavi, M., Azimi, S.A., Karati, S., Jaques, S.: DGSP: An Efficient Scalable Fully Dynamic Group Signature Scheme Using SPHINCS+. In: Daemen, J., Thomé, E. (eds.) Advances in Cryptology – EUROCRYPT 2026. pp. 454–485. Springer Nature Switzerland, Cham (2026)
12. Fadavi, M., Karati, S., Erfanian, A., Safavi-Naini, R.: DGMT: A fully dynamic group signature from symmetric-key primitives. Cryptology ePrint Archive, Paper 2024/1942 (2024), https://eprint.iacr.org.cn/2024/1942
13. Huelsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. RFC 8391 (May 2018). https://doi.org/10.17487/RFC8391, https://www.rfc-editor.org/info/rfc8391

14. Hülsing, A.: W-ots+ – shorter signatures for hash-based signature schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) Progress in Cryptology – AFRICACRYPT 2013. pp. 173–188. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
15. Libert, B., Ling, S., Mouhartem, F., Nguyen, K., Wang, H.: Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. Cryptology ePrint Archive, Paper 2016/101 (2016), https://eprint.iacr.org/2016/101
16. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. J. Cryptol. **36**(3) (May 2023). https://doi.org/10.1007/s00145-023-09470-6, https://doi.org/10.1007/s00145-023-09470-6
17. Ling, S., Nguyen, K., Wang, H., Xu, Y.: Constant-size group signatures from lattices. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography – PKC 2018. pp. 58–88. Springer International Publishing, Cham (2018)
18. 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)
19. Ling, S., Nguyen, K., Wang, H., Xu, Y.: Lattice-based group signatures: Achieving full dynamicity (and deniability) with ease. Theoretical Computer Science **783**, 71–94 (2019). https://doi.org/https://doi.org/10.1016/j.tcs.2019.03.023, https://www.sciencedirect.com/science/article/pii/S0304397519301884
20. National Institute of Standards and Technology (NIST): FIPS 205: Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication (FIPS) NIST FIPS 205 (2024), https://doi.org/10.6028/NIST.FIPS.205
21. del Pino, R., Lyubashevsky, V., Seiler, G.: Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 574–591. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243852, https://doi.org/10.1145/3243734.3243852
22. Yehia, M., AlTawy, R., Gulliver, T.A.: Gmmt: A revocable group merkle multi-tree signature scheme. In: Conti, M., Stevens, M., Krenn, S. (eds.) Cryptology and Network Security. pp. 136–157. Springer International Publishing, Cham (2021)

A Additional Security Arguments

This appendix provides supplementary details for the security analysis in the main body. The arguments are intentionally concise and are meant to clarify the reduction targets and the role of the credential-state assumptions.

A.1 Public-Verifier Anonymity

The public transcript of a PC-DGS signature consists of

$$(\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}, \sigma).$$

The WOTS+ public key pk is generated locally by the signer, but its association with a user identity is transmitted only through the confidential join/update

channel. The commitment $c = H(\mathsf{pk} \parallel r)$ hides the salt r and does not reveal uid. The index idx is allocated by the randomized pool independently of the user's identity and is therefore not a direct identifier.

The only identity-dependent value is $uidTag = H_{id}(uid)$, which is encrypted inside tok. A standard hybrid argument replaces the encryption of $H_{id}(u_0) \parallel \mathsf{epoch} \parallel \mathsf{idx} \parallel c$ with the encryption of $H_{id}(u_1) \parallel \mathsf{epoch} \parallel \mathsf{idx} \parallel c$. If a public adversary distinguishes the two hybrids with non-negligible probability, then the same distinguisher breaks IND-CCA confidentiality of AE. Hence the identity of the signer remains hidden from public verifiers.

A.2 Traceability Against Malicious Users

Traceability is considered against malicious users under an honest opening authority. Suppose an adversary outputs an accepted signature that cannot be opened. Since $\mathsf{GVerify}$ accepts, the certificate cert verifies as an XMSS certificate on

$$\mathsf{CertMsg} = \mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok}.$$

If such a credential was never issued, then cert is an XMSS forgery. Otherwise, the issuance record is indexed by $\mathsf{cid} = H(\mathsf{CertMsg})$, except with negligible probability of a collision in H.

If tok does not decrypt to the recorded tuple, then the adversary has produced a ciphertext that passes authentication with a modified plaintext, violating INT-CTXT security. If the token is valid, the honest opening authority retrieves the corresponding $\mathsf{TokenMap}$ entry and outputs a proof containing uid, r, and path. The judge then checks token consistency, the XMSS certificate, the WOTS+ signature, and the Merkle authorization path. Therefore, accepted signatures are traceable.

A.3 Non-Frameability

In the non-frameability game, the group administrator may be malicious and may know K_T. Therefore, token integrity is not the primary source of security against framing. The decisive checks are the user's non-substitutable Merkle authorization root and the WOTS+ signature under a user-generated public key.

A successful false opening to an honest user u must pass

$$c = H(\mathsf{pk} \parallel r), \quad \mathsf{MerkleVerify}(\mathsf{RegList}[u], c, \mathsf{path}) = 1.$$

If pk was not authorized by u, then the adversary must forge a valid membership witness for a commitment not included in the registered tree, breaking the binding of the Merkle authorization structure. If pk was authorized but the user did not sign, then the adversary must produce a valid WOTS+ signature under pk without the corresponding secret key, breaking WOTS+ one-time unforgeability. Thus, malicious-token generation alone does not enable framing, since the judge also enforces the registered-root and WOTS+ validity checks.

A.4 Credential-Reuse Detectability

Let Σ_1 and Σ_2 be generated with the same certified WOTS+ credential. Then both signatures contain identical credential fields:

$$(\mathsf{epoch}, \mathsf{idx}, \mathsf{pk}, c, \mathsf{tok}, \mathsf{cert}).$$

Consequently, both induce the same identifier

$$\mathsf{cid} = H(\mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok}).$$

Thus, any observer can detect reuse by comparing either the public credential tuple or cid. This property is not a double-spending prevention mechanism and does not require public verifiers to maintain a global signing list.

A.5 Signer-Side Forward Security

The signer-side forward-security property is local to the WOTS+ credential state. After signing, an honest user erases or irreversibly invalidates the used WOTS+ secret key and marks the credential as used. If the remaining local state is compromised later, the adversary obtains only unused credentials and public information associated with used credentials. Under the secure-erasure assumption, erased WOTS+ secret keys cannot be recovered, and therefore the adversary cannot generate additional signatures using past used credentials.

This property should not be confused with anonymity. An authorized opener or judge holding K_T can still open previous signatures. It also does not protect malicious users who intentionally back up WOTS+ secret keys before erasure.

A.6 Epoch Separation

For every certified credential, the epoch value is included in the token plaintext, the XMSS-certified message, the credential identifier, and the WOTS+ signed message:

$$\mathsf{TokenPlain} = \mathit{uidTag} \parallel \mathsf{epoch} \parallel \mathsf{idx} \parallel c,$$

$$\mathsf{CertMsg} = \mathsf{epoch} \parallel \mathsf{idx} \parallel \mathsf{pk} \parallel c \parallel \mathsf{tok},$$

$$\mathsf{cid} = H(\mathsf{CertMsg}), \quad \mathsf{SigMsg} = H(m \parallel \mathit{ctx} \parallel \mathsf{cid}).$$

Changing the epoch therefore invalidates at least one of the token consistency check, XMSS certificate verification, or WOTS+ message binding. Hence, a credential certified in one epoch cannot be migrated into another epoch.