Noname manuscript No.
(will be inserted by the editor)

DGMT: A Fully Dynamic Group Signature From Symmetric-key Primitives

Mojtaba Fadavi[a,1], **Sabyasachi Karati** [b,2], **Aylar Erfanian** [c,1], **Reihaneh Safavi-Naini** [d,1]

[1]Department of Computer Science, University of Calgary, Canada,
[2]Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata, India

Received: date / Accepted: date

Abstract A *group signatures* allows a user to sign a message anonymously on behalf of a group and provides accountability by using an *opening authority* who can "open" a signature and reveal the signer's identity. Group signatures have been widely used in privacy-preserving applications including anonymous attestation and anonymous authentication. *Fully dynamic* group signatures allow new members to join the group and existing members to be revoked if needed. *Symmetric-key based group signature schemes* are post-quantum group signatures whose security rely on the security of symmetric-key primitives such as cryptographic hash functions and pseudorandom functions.

In this paper, we design a symmetric-key based fully dynamic group signature scheme, called DGMT, that redesigns DGM (Buser et al. ESORICS 2019) and removes its two important shortcomings that limit its application in practice: (i) interaction with the group manager for signature verification, and (ii) the need for storing and managing an unacceptably large amount of data by the group manager. We prove security of DGMT (*unforgeability*, *anonymity*, and *traceability*) and give a full implementation of the system. Compared to all known post-quantum group signature schemes with the same security level, DGMT has the shortest signature size. We also analyze DGM signature revocation approach and show that despite its conceptual novelty, it has significant hidden costs that makes it much more costly than using traditional revocation list approach.

Keywords Symmetric cryptographic and hash functions · Privacy-preserving protocols · Group Signature Schemes · Post-quantum cryptography · Hash-based signature schemes

[a]e-mail: mojtaba.fadavi@ucalgary.ca
[b]e-mail: skarati@isical.ac.in
[c]e-mail:aylar.erfanianazadso@ucalgary.ca
[d]e-mail: rei@ucalgary.ca

1 Introduction

Digital signatures underpin trust and secure authentication and authorization over the internet. They have been used to construct certificates in public key infrastructure, credentials for user authentication and authorization, and signing transactions in electronic commerce and cryptocurrencies. Today's widely used digital signatures, such as (EC)DSA and RSA signature scheme, rely on the hardness of two computational problems, Discrete Logarithm Problem and Integer Factorization Problem, that can be efficiently solved by the Shor's quantum algorithm [49]. The prospect of the development of quantum computers, that potentially lead to the collapse of the cryptographic infrastructure of the Internet, has generated a flurry of research and development in academia and industry to design and develop post-quantum cryptographic systems, and has initiated standardization efforts around the world including NIST [1] and ETSI [22].

Post-quantum secure (PQ-secure) digital signature schemes provide security against adversaries with access to quantum computers. Known PQ-secure signature schemes have based their security on the hardness of computational problems for which no efficient quantum algorithm is known [4, 5, 16, 18, 28], or use only symmetric-key primitives such as hash functions and pseudorandom functions [10, 33, 34, 38, 44, 45]. Symmetric-key based approach is particularly attractive because, if designed securely, can only be threatened by advances in quantum algorithms that apply to unstructured data, such as Grover's algorithm [27].

Symmetric-key based signature schemes were first introduced in [38] as a *One-Time Signature (OTS)* scheme, using only hash functions and designed for signing a single message. Multi-message signature (MMS) schemes with short public key was proposed by Merkle by constructing a Merkle tree over a set of OTS public keys and using the root of the tree as the scheme's public key [45]. Merke tree approach and its variations reduce the size of the public key but increase the signature size, as the OTS public key and the authentication path from the leaf to the root must be included in the signature. *Winternitz One-Time Signature scheme (WOTS)* [45] is a particularly attractive OTS design in which the public key of the OTS can be derived from the OTS signature and so need not be included in the MMS signature, when using Merkle tree approach. Winternitz signature scheme and its variations serve as the main building block of stateful hash-based signature schemes that have been standardized by NIST, including XMSS [10, 15] and LMS [44].

Group Signature Schemes (GSS) [13] allow members of a group to sign on behalf of the group and provide anonymity and accountability for (group) members. In a GSS, a manager initializes the system and enrolls members to the group. A group member can sign anonymously on behalf of the group and their signature can be verified by the group public key. To provide accountability, signatures can be opened by the manager (or a separate opening authority) to reveal the identity of the signer. GSSs can be *static* where group membership is determined at the initialization time, *partially dynamic* that allow new members to join or existing members to be revoked after the initialization, and *fully dynamic* that support both joining new members and revoking existing members during the lifetime of the system [2, 3, 7]. GSSs and their variations have been used for Direct Anonymous Attestation in Trusted Platform Modules 1.2 [9], Enhanced Privacy Identification (EPID) [6], anonymous reputation systems [20], and digital rights management systems [36].

Symmetric-key based GSSs.

Group Merkle (GM) is the first symmetric-key based GSS. GM is a static GSS [19] and is based on a hash-based multi-message signature scheme in which the public key is the root of a Merkle tree. The tree is constructed over the public keys of a set of OTSs where each OTS (hash of the) public key forming a leaf node of the tree. The manager assigns a random subset of leaves to each user. The innovation of GM is that it constructs the random subset by applying a strong pseudorandom permutation to the leaves of the tree which hides the association of the users to the leaves and ensures the users' anonymity. The manager knows the secret key of the strong pseudorandom permutation and can reveal the identity of the signer, guaranteeing accountability. However, based on the GM's implementation, it supports 2^{18} signatures and can accommodate up to 2^{17} users as each user must be allocated at least two OTSs.

$\mathrm{GM}^{\mathrm{MT}}$*:* is a partially dynamic hash-based GSS [51] that uses GM approach and allows user revocation but not join. The scheme replaces the Merkle tree of GM with a multi-layer Merkle tree to increase the total number of signatures while keeping initialization time practical.

Dynamic GM (DGM) is a fully dynamic symmetric-key based GSS that extends GM approach to support user join and revoke operations [11]. DGM uses an IMT/SMT tree structure that consists of two types of Merkle trees: (i) An *Initial Merkle Tree (IMT)* that is built on a set of random strings that serve as its leaves, and (ii) a set of *Signing Merkle Trees (SMT)*, each built on the public keys of a set of OTSs. The root of each SMT is linked to an internal node of the IMT, referred to as a *fallback node*, using a *fallback key* that is part of the signature [1]. See Figure 3. The number of fallback keys increases with the number of OTSs that DGM provides. The manager generates and stores fallback keys as new users join the group. During verification, the verifier queries the validity of a fallback key, that is part of the signature, by interacting with the manager. This IMT/SMT structure allows the manager to gradually construct the tree over the system's lifetime and avoid the high computation cost of building the entire tree during the initialization phase. This, however, requires interaction with the group manager for verification. See Subsection 3.2 for more details on DGM.

DGM permits multiple SMTs per fallback node and so allows the total number of signatures that is provided by the system to grow. This, however, not formalized and included in the security proof. DGM also uses an innovative approach to user revocation that employs a *Symmetric Puncturable Encryption scheme (SPE)*. In SPEs the decryption key can be updated and punctured on a desired input to prevent decryption of a ciphertext corresponding to that input. Revocation in DGM uses encryption and decryption functions of SPE to determine whether a signature is revoked or not. See Subsection 2.1 and Section 6 for more details on SPE and SPE-based revocation of DGM, respectively.

DGM shortcomings. DGM has two important shortcomings: (i) It requires interaction between the verifier and the manager for every signature verification, requiring the manager to be online continuously to respond to the verifiers' queries on the validity of the fallback key which is a part of the signature; (ii) It requires the manager to store the index of each OTS assigned to every user in order to open signatures and revoke users, leading to a storage size that grows linearly with the total number of signatures generated by the scheme, denoted by $\mathsf{T}_{\mathsf{tot}}$. Additionally, the manager must store all fallback keys to verify them for users. For instance, to support $\mathsf{T}_{\mathsf{tot}} = 2^{64}$ OTSs, the manager would need to store 2^{64} leaf node indexes for signature opening and revocation, along with 2^{62} fallback keys, amounting to approximately $10^{8.7}$ terabytes of storage [51]. This storage demand is unacceptably large, making it impractical for real-world applications.

Another less obvious drawback of DGM that is uncovered by our work is the very inefficient approach to revoca-

[1] We use the terminology from [11], particularly for terms like 'fallback node' and 'fallback key.'

tion. DGM SPE-based approach to revocation although appears novel, but incurs significant cost. See Section 6.

1.1 Our Contributions

In this work, we propose a symmetric-key based fully dynamic group signature scheme, called DGMT, that addresses the design shortcomings of DGM, and implement and evaluate its performance. Our main contributions are as follows.

1- A new tree structure. In all existing symmetric-key based GSSs discussed above, a *virtual* tree is constructed on a set of leaves that each corresponds to the public key of an OTS. The root of this tree is the GSS's public key. The innovation of each system is in the way the tree leaves are managed during assignment and revocation, with the goal of providing user anonymity and accountability, and security and efficiency for the GSS. We construct a virtual tree on the set of OTSs that extends the IMT/SMT structure of DGM to $\mathrm{IMT}/\mathrm{SMT}^{\mathrm{MT}}$ and explicitly uses multiple $\mathrm{SMT}^{\mathrm{MT}}$ per each IMT tree internal node. The tree is designed to accommodates the required total number of signatures, $\mathsf{T}_{\mathsf{tot}}$, using a relatively small number of fallback keys. This allows the list of fallback keys can be generated and published during initialization (e.g. on a public bulletin board). The verifier can directly verify a signature by accessing the published information and without the need to interact with the manager.

2- Reducing server storage. We design a new signature assignment algorithm where the manager's storage is proportional to the *maximum number of users*, denoted by $\mathsf{N}_{\mathsf{max}}$, rather than the total number of signatures $\mathsf{T}_{\mathsf{tot}}$. This significantly reduces the manager's storage requirements, as each user is typically associated with a large number of OTSs. To achieve this, we assign a fixed, unique (and disjoint) interval of the list $[0, 1, \cdots, \alpha - 1]$ for an integer α, to each user, and use a strong pseudorandom permutation to map the intervals to (pseudo)random leaves of the signature tree.

3- Implementation. We provide a full implementation of DGMT and all its algorithms. The software is written in C language, which is the NIST-recommended reference language, and the code is available at `https://github.com/submissionOfCode/DGMT_ref`. In addition to signature and verification algorithms, we have implemented key management algorithms, that consist of the initial key assignment, and user join and revocation. Key generation, and storage that support signature opening, and management of revocation list are the most complex parts of our code. (DGM only provides implementation of the signing and verification algorithms.)

For user revocation, DGMT uses a public revocation list. In Section 6, we analyse the user revocation approach of DGM that is based on SPE, and compare it with DGMT approach, demonstrating that DGM's approach is significantly more costly in both computation and storage cost. Table 8 summarizes our results. In Appendix A, we show an alternative way of using SPE-based revocation in DGM which improves its efficiency and reduces the cost of revocation.

1.2 Related work

To provide combined anonymity and accountability, symmetric-key based group signature schemes have used two approaches. The schemes in [11, 19, 51, 52] use multi-time hash-based signature schemes together with other symmetric-key cryptographic primitives such as pseudo-random permutations and symmetric-key encryption. The group public key is the root of a Markle tree that is constructed over a set of leaves where each leaf corresponds to an OTS public key. The group manager allocates a "random" subset of leaves to each user and provides them with the secret keys of the allocated OTSs and the corresponding authentication paths to the root of the Merkle tree. The user uses one OTS and its associated authentication path at most once. DGM and $\mathrm{GM}^{\mathrm{MT}}$ allow the user to request additional OTS keys from the group manager by presenting their secret credentials, that are obtained during enrollment, to authenticate themselves. The schemes in [6, 14, 31] also use symmetric-key primitives and Merkle tree but employ Non-Interactive Zero-Knowledge (NIZK) proofs to construct the signature. We note that the scheme in [6] is an EPID and does not provide opening function to link a signature to a user, and the scheme in [31] is a static group signature. The scheme in [14] however is a fully dynamic symmetric-key based group signature that uses a hybrid approach: it uses a multi-time hash-based signature scheme, that is a modified version of $\mathrm{SPHINCS}^{+}$, and is optimized for multiparty computation, and a NIZK to achieve anonymity. In Subsection 4.1 we give a comparison of DGM and DGMT algorithms, and in Section 1.2.1 give a more detailed look at [14] and its comparison with DGMT.

Table 1 summarizes comparison of all known symmetric-key based group signature schemes. All rows except the last row of the Table 1 are sourced from Table 1 in [14].

1.2.1 Sphinx-in-the-Head

Sphinx-in-the-Head (SiTH) is a post-quantum fully-dynamic symmetric-key based group signature scheme. It uses a new mult-message hash-based signature scheme which is a modification of $\mathrm{SPHINCS}^{+}$ with the goal of making it more "friendly" for MPC (multiparty computation) that is needed for NIZK that is used in the scheme.

SiTH splits the role of the manager between a *group issuer* and a *group tracer* each receiving a corresponding

Table 1 A Comparison of Hash-Based Group Signature Schemes

Schemes	Underlying Signatures	Group Credentials	Static/Fully Dynamic	Group Size*	Non-frameability
G-Merkle [19]	OTS	Merkle signature	static	2^{6**}	no
DGM [11]	OTS	Merkle signature	Fully dynamic	–	no
DGM$^+$ [52]	OTS	XMSS-T	Fully dynamic	–	no
GM^{MT} [51]	OTS	XMSS-T	dynamic	2^{16***}	no
KKW [31]	NIZK	Merkle signature	static	2^{13}	no
BEF [6]	NIZK	Merkle or Goldreich signature	static	–	no
SiTH [14]	NIZK	F-SPHINCS$^+$	Fully dynamic	2^{60}	yes
This work	OTS	Merkle signature	Fully dynamic	2^{15}	no

* refers to the maximum group size that has been implemented and reported in the paper; "–" indicates that no implementation has been reported.
** To ensure the anonymity of GM, each user requires two OTSs. Therefore, according to the GM's implementation in Table 1, if the GM provides 2^{18} OTSs, it can support up to 2^{17} users.
*** In Table 4 in [51], it is mentioned at most 2^{10}.

private key from the user, preventing each of the two authorities to "frame" the user by generating a signature on their behalf, and hence providing *non-frameability*. This makes SiTH unique among other post-quantum symmetric-key based full-dynamic group signature schemes that use a single trusted group manager and do not provide non-frameability. SiTH scheme uses NIZK that allows the user to prove to the verifier a set of relations that are defined by its secret values, the signed message, and the published values of the system. Using NIZK results in a much slower signature generation and verification time, and a significantly larger signature size, compared to DGMT.

Similar to DGMT, SiTH uses a revocation list to keep track of the revoked users. The revocation list will include the secret tracing key of the user and so revokes all the signatures of the user, including the past ones. Authors noted that SiTH can be made forward anonymous by including the hash of the private tracing key in the revocation list. This however increases the computation cost of the signer who must show that the signing key has not been revoked.

In DGMT however the signatures are individually revoked. In our revocation algorithm we will include all signatures of a revoked user to the revocation list and so effectively revoke all signatures of the user. DGMT, however, can provide forward anonymity by revoking the signatures individually. This is by including the output of the strong pseudorandom permutation on the index of the revoked OTSs and all the remaining keys of the user, leaving out its past signatures. Thus will require minor changes to the current algorithms (e.g. "revoked" flag) that will not affect security.

1.2.2 DGMT and Other Post-Quantum GSSs

There is a large body of research on *Lattice-based* GSSs including static [4, 21, 26, 37], partially dynamic [39, 40, 42], and fully dynamic lattice-based GSSs [41]. There are also constructions of code-based [23, 46] and isogeny-based [4] GSS. Table 2 gives a comparison of signature sizes of the most efficient known post-quantum GSSs. Rows 3–5 of the table are from Table 1 in [4]. All schemes except DGMT and SiTH are static. The only known fully dynamic GSS that uses a computational assumption is lattice-based [41], and is not included in the table because the signature size is only known asymptotically, that is $\tilde{O}(\lambda \log N)$ bits, where λ and N are the security parameter and group size, respectively. As seen in the table, DGMT has the shortest signature size among all known schemes.

1.3 Organization

Section 2 is preliminaries and Section 3 introduces a security model for fully dynamic symmetric-key based GSS. Section 4 gives the construction of DGMT, followed by the presentation of its security proof in Section 5. An analysis of the revocation using SPE in DGM is provided in Section 6. Section 7 is the implementation of DGMT and its experimental results. Appendix A gives a more efficient SPE-based revocation method for DGM, and Appendix B outlines an approach for reducing setup time in DGMT.

1.4 Notations

Let $\mathbb{N}$ be the set of positive integers and $\lambda \in \mathbb{N}$ be the security parameter. For two given integers $a, b \in \mathbb{N}$, we denote the representation of a in base b by $(a)_b$, and for $a \le b$, we denote the set $\{x \in \mathbb{Z} \mid a \le x \le b\}$ by $[a,b]$. For a given list L, we denote the n-th element of this list by $L[n]$. For two given strings x and y, we denote their concatenation by $x \parallel y$. If S is a set or a list, we denote its size by $|S|$ and the operation of randomly selecting an element x from S by $x \overset{\$}{\leftarrow} S$. We write

Table 2 Signature size of the most efficient post-quantum GSSs, given in Kilobyte (KB). The signature sizes of the GSSs [4, 14, 21, 23] depend on the number of users (group size), denoted by N. To show how many users are accommodated in each scheme, we consider $N \in \{2^4, 2^5, 2^{10}, 2^{12}, 2^{15}, 2^{20}, 2^{21}, 2^{24}, 2^{60}\}$.

GSS	Signature Type	Based on	bit security	N								
				2^4	2^5	2^{10}	2^{12}	2^{15}	2^{20}	2^{21}	2^{24}	2^{60}
[23]	Static	Code	80	157	–	–	205	–	–	–	200704	–
[43]	Static	Lattice	–	203								
[4]	Static	Lattice	–	–	126	129	–	–	–	134	–	–
[21]	Static	Lattice	–	–	12	19	–	–	–	–	–	–
[4]	Static	Isogeny	128	–	6	9	–	–	–	15.5	–	–
[14]	Fully dynamic	Symmetric	128	–	–	571	–	–	851	–	–	2000
DGMT	Fully dynamic	Symmetric	128	5.75*					–	–	–	–
DGMT	Fully dynamic	Symmetric	192	11.62*					–	–	–	–

* See the Subsections 4.4 for computing signature size and the bit security of DGMT.
"–" shows that either scheme cannot accommodate N users or no information has been reported in their papers.

$x \leftarrow A(a)$ for an algorithm A that runs on input a and outputs x. We call a function $\varepsilon(\cdot) : \mathbb{N} \to \mathbb{R}^+$ negligible in the security parameter λ if for every polynomial $p(\cdot)$ and all sufficiently large values of λ, $\varepsilon(\lambda) < \frac{1}{p(\lambda)}$ holds. $\lceil \cdot \rceil$ is the ceiling function. For a given adversary $\mathcal{A}$ and $i \in \mathbb{N}$, $\mathcal{A}^{f_1,\cdots,f_i}$, indicates that the adversary $\mathcal{A}$ is given access to the oracles $f_1, \cdots, f_i$.

2 Preliminaries

In the following, we recall some cryptographic primitives [24, 32, 47, 48]. Let $m(x)$ and $\ell(x)$ be two polynomials and $\lambda \in \mathbb{N}$ be the security parameter.

One-way Function (OWF): A function $f : \{0,1\}^* \to \{0,1\}^*$ is *one-way* if (i) for all $x \in \{0,1\}^*$, $f(x)$ is efficiently computable, and (ii) it is hard to invert the function; that is for every *Probabilistic Polynomial Time (PPT)* adversary $\mathcal{A}$

$$\Pr[x \overset{\$}{\leftarrow} \{0,1\}^\lambda \mid f(\mathcal{A}(1^\lambda, f(x))) = f(x)]$$

is negligible in terms of λ.

Pseudorandom Function (PRF): A functions

$$F : \{0,1\}^\lambda \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{\ell(\lambda)}$$

is *pseudorandom* if (i) for all k and x, $F(k,x)$ is efficiently computable, and (ii) for any PPT adversary $\mathcal{A}$

$$Adv_{\mathcal{A}}^{\mathsf{PRF}} = \left| \Pr\left[\mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1\right] \right|$$

is negligible in terms of λ, where $k \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and f is chosen randomly from the the set of all functions mapping $m(\lambda)$ bits to $\ell(\lambda)$ bits.

Strong Pseudorandom Permutation (SPRP): A function

$$\mathsf{E} : \{0,1\}^\lambda \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{m(\lambda)}$$

is *a strong pseudorandom permutation* if (i) for all k and x, $\mathsf{E}(k,x)$ and $\mathsf{E}^{-1}(k,x)$ is efficiently computable, where $\mathsf{E}^{-1} : \{0,1\}^\lambda \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{m(\lambda)}$, and (ii) for any PPT adversary $\mathcal{A}$

$$Adv_{\mathcal{A}}^{\mathsf{SPRP}} = \left| \Pr\left[\mathcal{A}^{\mathsf{E}(k,\cdot),\mathsf{E}^{-1}(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)}(1^\lambda) = 1\right] \right|$$

is negligible in terms of λ, where $k \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and π is chosen randomly from the the set of all permutations on $\{0,1\}^{m(\lambda)}$.

Collision-resistant Hash Function (CRH): A set of functions

$$\mathcal{HF} = \{\mathsf{H}_k : \{0,1\}^{m(\lambda)} \to \{0,1\}^\lambda\}_{k \in \{0,1\}^\lambda, m(\lambda) > \lambda},$$

is a family of collision-resistant hash functions if (i) for all key k and $x \in \{0,1\}^{m(\lambda)}$, H_k is efficiently computable, (ii) for all PPT adversaries $\mathcal{A}$

$$Adv_{\mathcal{A},\mathcal{HF}}^{\mathsf{col}} = \Pr[k \overset{\$}{\leftarrow} \{0,1\}^\lambda,\ (x,x') \leftarrow \mathcal{A}(k) \mid x \neq x', \text{ and } \mathsf{H}_k(x) = \mathsf{H}_k(x')]$$

is negligible in terms of λ.

Symmetric Key Encryption: A symmetric key encryption scheme is a tuple of polynomial-time algorithms $\mathcal{SE} = (\mathsf{SE.KG}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ where the key generation algorithm $\mathsf{SE.KG}$ generates a random λ-bit key $k \leftarrow \mathsf{SE.KG}(1^\lambda)$, and the encryption and decryption algorithms work as follows. The encryption algorithm $\mathsf{SE.Enc} : \{0,1\}^\lambda \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{m(\lambda)}$ encrypts a message m into the ciphertext c as $c \leftarrow \mathsf{SE.Enc}(k,m)$, and the decryption algorithm $\mathsf{SE.Dec} : \{0,1\}^\lambda \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{m(\lambda)}$ decrypts a ciphertext c into the plaintext m as $m \leftarrow \mathsf{SE.Dec}(k,c)$. The correctness property of $\mathcal{SE}$ says that $\mathsf{SE.Dec}(k, \mathsf{SE.Enc}(k,m)) = m$.

2.1 Symmetric Puncturable Encryption Scheme (SPE)

DGM uses an SPE-based revocation scheme. To provide a concrete analysis of DGM revocation algorithm, we recall the SPE algorithm that was proposed in [50] and used in DGM. The construction uses the *puncturable pseudorandom function (Pun-PRF)* given in [29].

Let $\mathcal{K}$, $\mathcal{K}_P$, $\mathcal{Y}$ and $\mathcal{T}$ be the set of key space, punctured key space, output space, and tag space (or input) of the Pun-PRF, respectively. Let F' be a PRF, $F' : \mathcal{K} \times \mathcal{T} \to \mathcal{Y}$. A *Pun-PRF F*, constructed from F' is a PRF with a pair of algorithms $(\mathsf{F.Punc}, \mathsf{F.Eval})$ defined as follows: $\mathsf{F.Punc}$ takes a key $k \in \mathcal{K}$ and a tag $t \in \mathcal{T}$, and outputs a *punctured key* $pk_t \in \mathcal{K}_P$. The evaluation function $\mathsf{F.Eval}$ takes a punctured key pk_t and a tag t', computes $\mathsf{F.Eval}(k,t') = F'(k,t')$ if pk_t is not punctured at tag t', and outputs fail $(\perp)$, otherwise. That is,

$$\mathsf{F.Punc} : \mathcal{K} \times \mathcal{T} \to \mathcal{K}_P, \text{ s. t. } \mathsf{F.Punc}(k,t) = pk_t$$

$$\mathsf{F.Eval} : \mathcal{K}_P \times \mathcal{T} \to \mathcal{Y} \cup \{\perp\}, \text{ s.t.}$$

$$\mathsf{F.Eval}(pk_t,t') = \begin{cases} F'(k,t') = y, & \text{if } t \neq t', \\ \perp, & \text{if } t = t'. \end{cases}$$

A Construction for SPE. A symmetric d-puncturable encryption scheme is a tuple of algorithms

$$\mathcal{SPE} = (\mathsf{SPE.KeyGen}, \mathsf{SPE.Enc}, \mathsf{SPE.Punc}, \mathsf{SPE.Dec})$$

that allows the input key to be punctured on up to d tags. The $\mathcal{SPE}$ master key msk remains unchanged. The encryption key is computed once but the decryption key is updated with each puncturing of a tag. The construction in [50] uses three building blocks: (i) a family of collision-resistant hash functions $\{\mathsf{H}_k : \mathbb{N} \to \mathcal{K}\}_{k\in\mathcal{K}}$ (ii) a symmetric key encryption system $(\mathsf{SE.Enc}, \mathsf{SE.Dec})$ with key space $\mathcal{K}$ and (iii) a PRF F' as defined above.

The initial decryption key of $\mathcal{SPE}$ is the same as the encryption key msk, that is $SK_0 = \{msk\}$. The decryption key, however, is updated with each punctured tag t'_i. $\mathcal{SPE}$ algorithms are outlined below.

- $msk \leftarrow \mathsf{SPE.KeyGen}(1^\lambda, d)$: The algorithm takes the security parameter λ and $d \in \mathbb{N}$ and outputs the master encryption key $msk = (sk_0, d)$, where $sk_0 \xleftarrow{\$} \mathcal{K}$.
- $ct_t \leftarrow \mathsf{SPE.Enc}(msk,m,t)$: The algorithm takes the encryption key msk, message m and tag t, and outputs the ciphertext ct_t.

  $$\{sk_i : 1 \le i \le d,\ \ sk_i \leftarrow \mathsf{H}_{sk_{i-1}}(i)\},$$
  $$\kappa_t \leftarrow \bigoplus_{i=0}^{d} F'(sk_i,t),$$
  $$ct_t \leftarrow \mathsf{SE.Enc}(\kappa_t, m).$$
- The Decryption algorithm uses a punctured key, a ciphertext, and a tag as inputs and outputs a ciphertext or failure. First, we describe how $\mathsf{SPE.Punc}$ updates the decryption key, and then how to decrypt the ciphertext.
  1. $SK_i \leftarrow \mathsf{SPE.Punc}(SK_{i-1}, t'_i)$: The algorithm updates the decryption key for the i-th punctured tag, given the previous key and the associated set of punctured tags. It takes $SK_{i-1} = \{msk_{i-1}, psk_1, \cdots, psk_{i-1}\}$, where $msk_{i-1} = (sk_{i-1}, d)$ for $1 \le i \le d$ and $SK_0 = msk$, for the set of tags $T_{i-1} = \{t'_1, t'_2, \cdots, t'_{i-1}\}$, and outputs SK_i. $\mathsf{F.Punc}$ is defined recursively below.

     $$psk_i \leftarrow \mathsf{F.Punc}(sk_{i-1}, t'_i),$$
     $$sk_i \leftarrow \mathsf{H}_{sk_{i-1}}(i),$$
     $$msk_i = (sk_i, d)$$
     $$SK_i = \{msk_i, psk_1, \cdots, psk_i\}.$$
  2. $m_t \leftarrow \mathsf{SPE.Dec}(SK_i, ct_t, t)$: The algorithm takes the SK_i, the ciphertext ct_t and a tag t, and outputs the plaintext m_t as follows. If $1 \le i < d$, compute $sk_j \leftarrow \mathsf{H}_{sk_{j-1}}(j)$ for $i < j \le d$.

     $$\kappa'_t \leftarrow \bigoplus_{j=1}^{i} \mathsf{F.Eval}(psk_i, t) \bigoplus_{j=i}^{d} F'(sk_i, t),$$
     $$m_t \leftarrow \mathsf{SE.Dec}(\kappa'_t, ct_t)$$

2.2 One-Time Signature (OTS)

An OTS scheme is a digital signature scheme that is designed for signing a single message. It consists of three polynomial-time algorithms $\mathcal{OTS} = (\mathsf{OTS.KG}, \mathsf{OTS.Sig}, \mathsf{OTS.Vf})$, which operate as follows

- $(\mathsf{OTS.sk}, \mathsf{OTS.pk}) \leftarrow \mathsf{OTS.KG}(1^\lambda)$. This algorithm generates a key pair $(\mathsf{OTS.sk}, \mathsf{OTS.pk})$, where $\mathsf{OTS.sk}$ is the secret key and $\mathsf{OTS.pk}$ is the public key.
- $\sigma \leftarrow \mathsf{OTS.Sig}(\mathsf{OTS.sk}, m)$. This algorithm signs a message m with a secret key $\mathsf{OTS.sk}$ and outputs a valid digital signature σ.
- $0/1 \leftarrow \mathsf{OTS.Vf}(m, \sigma, \mathsf{OTS.pk})$. This is a deterministic algorithm that checks the validity of the signature σ on message m using $\mathsf{OTS.pk}$. $\mathsf{OTS.Vf}$ outputs 1 if the message and signature pair is valid, otherwise, it outputs 0.

An OTS scheme is called *existentially unforgeable under chosen message attack (EU-CMA)* if for any PPT adversary $\mathcal{A}$

$$\begin{aligned} Adv^{\mathsf{EU-CMA}}_{\mathcal{A},\mathcal{OTS}} = \Pr\big[&(\mathsf{OTS.sk}, \mathsf{OTS.pk}) \leftarrow \mathsf{OTS.KG}(1^\lambda);\\ &(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{OTS.Sig}(\mathsf{OTS.sk},\cdot)}(\mathsf{OTS.pk});\\ \text{Let } (m,\sigma) \text{ be the query-answer pair of } &\mathsf{OTS.Sig}(\mathsf{OTS.sk},\cdot) \mid\\ &m \neq m^* \wedge 1 \leftarrow \mathsf{OTS.Vf}(m^*, \sigma^*, \mathsf{OTS.pk})\big] \end{aligned}$$

is negligible in terms of λ [33].

Winternitz One-time Signature (WOTS) is one of the most efficient OTS schemes, designed for both lightweight performance and strong security.

Let $f : \{0,1\}^* \to \{0,1\}^\lambda$ be a one-way function, H be a hash function chosen randomly from the collision-resistant hash function family $\mathcal{HF}$[2], and f^i be the i-fold iteration function of f for every $i \in \mathbb{N}$, i.e. for all $x \in \{0,1\}^*$ if $i > 0$, $f^i(x) = f(f^{i-1}(x))$, and $f^0(x) = x$. WOTS relies on two key parameters: the security parameter λ and the Winternitz parameter w. These parameters are used to define the following quantities:

$$\xi_1 = \lceil \frac{\lambda}{w} \rceil, \qquad \xi_2 = \lfloor \frac{\log(\xi_1(2^w - 1))}{w} \rfloor + 1, \qquad \xi = \xi_1 + \xi_2.$$

WOTS consists of three algorithms

$$\mathcal{WOTS} = (\mathsf{WOTS.KG}, \mathsf{WOTS.Sig}, \mathsf{WOTS.Vf}),$$

and works as follows.

$(\mathsf{WOTS.sk}, \mathsf{WOTS.pk}) \leftarrow \mathsf{WOTS.KG}(1^\lambda)$: The key generation algorithm produces a secret key $\mathsf{WOTS.sk} = (x_0, x_1, \ldots, x_{\xi-1})$, where $x_i \xleftarrow{\$} \{0,1\}^\lambda$ for all $0 \leq i \leq \xi - 1$, and computes the public key $\mathsf{WOTS.pk} = (y_0, y_1, \ldots, y_{\xi-1})$, where $y_i = f^{2^w-1}(x_i)$ for all $0 \leq i \leq \xi - 1$.

$\mathsf{WOTS}.\sigma \leftarrow \mathsf{WOTS.Sig}(\mathsf{WOTS.sk}, m)$: For a given message m, this sginng algorithm first computes $d \leftarrow \mathsf{H}(m)$ and represents d in base 2^w as $(d)_{2^w} = (b_0, \cdots, b_{\xi_1 - 1})_{2^w}$. Then, it computes the checksum $c = \sum_{i=0}^{\xi_1 - 1}(2^w - 1 - b_i)$, and appends $(c)_{2^w}$ to $(d)_{2^w}$ to obtain

$$B = (d)_{2^w} \parallel (c)_{2^w} = (b_0, \cdots, b_{\xi_1 - 1}, b_{\xi_1}, b_{\xi_1 + 1}, \cdots, b_{\xi - 1})_{2^w}.$$

The signature on message m is $\mathsf{WOTS}.\sigma = (\sigma_0, \cdots, \sigma_{\xi-1})$, where

$$\sigma_i = f^{b_i}(x_i), \qquad \text{for all} \quad 0 \leq i \leq \xi - 1.$$

$0/1 \leftarrow \mathsf{WOTS.Vf}(m, \mathsf{WOTS}.\sigma, \mathsf{WOTS.pk})$: The verification algorithm computes $B = (d)_{2^w} \parallel (c)_{2^w}$ using the message m as done in the signing algorithm $\mathsf{WOTS.Sig}$. It verifies the signature $\mathsf{WOTS}.\sigma$ by outputting 1 if

$$\mathsf{WOTS.pk} = \left(f^{2^w - 1 - b_0}(\sigma_0), \ldots, f^{2^w - 1 - b_{\xi-1}}(\sigma_{\xi-1})\right),$$

otherwise, it outputs 0.

2.3 Merkle Tree and Multi-message Signature Scheme (MSS)

A Merkle tree is a binary tree in which every leaf node is the hash value of a data block, and each non-leaf node is the hash value of the concatenation of its two children. The root of the Merkle tree, also known as the Merkle root, serves as a compact representation of the entire dataset, that can be used to provide membership proof for the dataset elements. This structure efficiently verifies the integrity of any data block using a logarithmic number of hash operations and a comparison.

Figure 1 gives an example of a Merkle tree of height 3 constructed on 8 data blocks. The leaf nodes $y_3[i]$, $0 \leq i \leq 7$, are the hash values of the data blocks and all the other nodes are computed as $y_j[i] \leftarrow \mathsf{H}(y_{j+1}[2i] \parallel y_{j+1}[2i+1])$, where $0 \leq j \leq 2$, $0 \leq i < 2^j$, and H is chosen randomly from the collision-resistant hash function family $\mathcal{HF}$. To verify the membership of a data block, say $y_3[3]$ (marked in orange), we provide the authentication path that is the sibling nodes of the nodes on the path from the mentioned data block to the root. For example the authentication path for $y_3[3]$ is the set $\{y_3[2], y_2[0], y_1[1]\}$ (marked in yellow). Now we can sequentially compute the nodes $y_2[1], y_1[0]$ and $y_0[0]$ (marked in green) which are on the path from $y_3[3]$ to root $y_0[0]$. By comparing the computed root and published root, we can determine the membership of a data block.

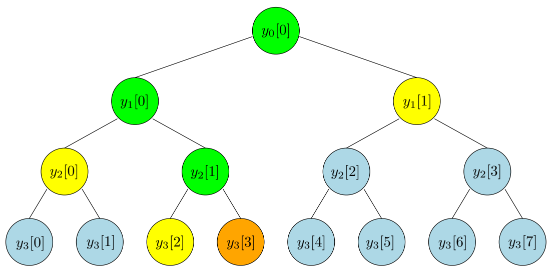

Fig. 1 A Merkle tree of height 3

Multimessage signaure scheme (MSS). To construct an MSS that supports up to $\mathsf{T_{tot}} = 2^h$ signatures and has a short public key, Merkle [45] proposed to construct a Merkle tree on the hash values of the public keys of the set OTSs that are used in the signature, and use the root of the tree as the MSS public key. Specifically, for $0 \leq i < 2^h$, the i-th leaf node is defined as $y_h[i] \leftarrow \mathsf{H}(\mathsf{OTS.pk}_i)$, where $(\mathsf{OTS.sk}_i, \mathsf{OTS.pk}_i)$ is the i-th OTS key pair. For $0 \leq j < h$ and $0 \leq i < 2^j$, the i-th node at height j is computed as $y_j[i] \leftarrow \mathsf{H}(y_{j+1}[2i] \parallel y_{j+1}[2i+1])$. The root of the Merkle tree, $y_0[0]$, acts as the public key of the scheme.

MSS uses leaf nodes of the Merkle tree to sign messages. The signature of a message m, using the i-th OTS is $\sigma = (i, \mathsf{OTS}.\sigma_i, \mathsf{OTS.pk}_i, A.path_i)$, where $\mathsf{OTS}.\sigma_i \leftarrow \mathsf{OTS.Sig}(\mathsf{OTS.sk}_i, m)$, and $A.path_i = (a_0, a_1, \cdots, a_{h-1})$ is authentication path for the i-leaf node, that is $y_h[i]$. To verify the signature σ on the message m, the verifier first verifies the $\mathsf{OTS}.\sigma_i$ by $\mathsf{OTS.Vf}(m, \mathsf{OTS}.\sigma_i, \mathsf{OTS.pk}_i)$ and upon success they compute the root of the Merkle tree using the authentication path $A.path_i$ and $y_h[i]$. If the computed root is the same as the given public key $y_0[0]$, signature σ_i will be accepted.

[2]From now on, for simplicity, we only write H rather than H_k.

Remark 1 Using WOTS in a multi-message signature scheme removes the need for including the public key of the OTS in the signature. The signature of message m using the WOTS key pair $(\mathsf{WOTS.sk}_i, \mathsf{WOTS.pk}_i)$ is $\sigma = (i, \mathsf{WOTS}.\sigma_i, A.path_i)$, where $\mathsf{WOTS}.\sigma_i = (\sigma_0, \cdots, \sigma_{\xi-1}) \leftarrow \mathsf{WOTS.Sig}(\mathsf{WOTS.sk}_i, m)$. To verify the message-signature pair (m, σ), the verifier computes $\left(f^{2^w-1-b_0}(\sigma_0), \ldots, f^{2^w-1-b_{\xi-1}}(\sigma_{\xi-1})\right)$ to derive the corresponding leaf node $y_h[i]$, and verifies if the computed leaf node and the authentication path $A.path_i$ matches the root of the Merkle tree, $y_0[0]$. If the match succeeds the signature σ is accepted.

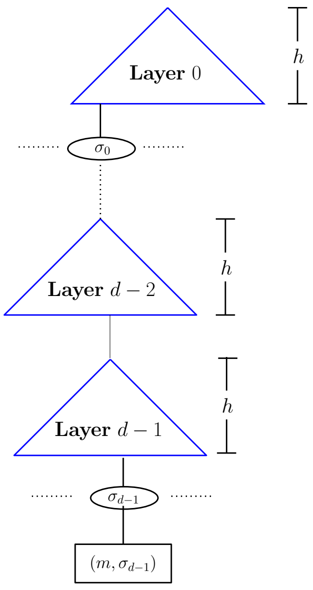

Fig. 2 A d-layer Merkle tree

Multi-tree structure. As we discussed above, constructing an MSS using a Merkle tree requires deriving the public key by computing all the leaves of the tree first. For a large number $\mathsf{T}_{\mathsf{tot}}$, this would require a significant amount of initial computation and storage. A d-layer Merkle tree reduces the initial computation required for a large number $\mathsf{T}_{\mathsf{tot}}$ by constructing the tree gradually [35].

An overview of a d-layer Merkle tree is shown in Figure 2. The root of the Layer 0 Merkle tree is the public key of the signature scheme and to compute the public key, only the topmost layer of the multi-tree has to be constructed. Every leaf node of a Merkle tree of layer $d-1$ signs a message, and every leaf node of the sub-trees of the layer $0 \leq i < d-1$ signs the Merkle root of a sub-tree of the layer below, like σ_{d-1} and σ_0 in Figure 2, respectively. Thus, each signature of MSS in a multi-tree structure contains d OTS signatures, one for each layer. Consequently, verifying an MSS signature in a multi-tree structure includes the cost of verifying d OTS signatures.

3 Fully Dynamic Group Signature

In this section, we describe a model of a fully dynamic group signature ($\mathcal{FDGS}$). Our model is based on [3, 7] and is adapted to fully dynamic symmetric-key based GSS [11]. Similar to [11] (i) we consider a single trusted authority for key generation and group management (i.e. join and revoke), as well as opening of signatures, and (ii) include a *request subroutine* that allows users to request new keys when their existing keys are run out.

Our security model is based on [3, 7, 12] and loosely follows [11]. In particular, we adopt the anonymity notion that was introduced in [12] and do not allow the adversary to corrupt the manager or the two selected honest users in the anonymity game.

A $\mathcal{FDGS}$ is composed of three types of entities:

– A *Trusted manager*, denoted by $\mathcal{M}$, is the central authority responsible for the perfect functioning of the group. $\mathcal{M}$ allows new members to join the group, can reveal the identity of a signer, and can revoke the signing ability of misbehaved members and their generated signatures.
– *Members/users* anonymously sign messages on behalf of the group.
– *Verifier* verifies the validity of a group signature using only the public parameters.

Let λ be the security parameter and $\mathsf{FDGS.SetPr}$ be the *setup parameters*. A $\mathcal{FDGS}$ consists of the following polynomial-time algorithms.

– $(\mathsf{FDGS.SK}, \mathsf{FDGS.PubPr}) \leftarrow \mathsf{FDGS.KG}(1^\lambda, \mathsf{FDGS.SetPr})$: The manager $\mathcal{M}$ runs this algorithm to generate the *manager's secret key* $\mathsf{FDGS.SK}$, including the *manager master secret key* msk and other group secret information, and the *public parameters* $\mathsf{FDGS.PubPr}$, including the *group public key* gpk and other group public information that are necessary for verifying signatures.
– $((\mathsf{PL}_{\mathcal{M}}, \mathsf{ID}), (\mathsf{id}, c_{\mathsf{id}}))/\perp \leftarrow \mathsf{FDGS.Join}(\mathsf{Username})$: This is an interactive joining protocol between a user with identity $\mathsf{Username}$, and the manager $\mathcal{M}$. In line with the model in [3], the communication occurs over secure (i.e., private and authenticated) channels, with the user initiating the protocol by sending their identity $\mathsf{Username}$. Let ID represent the list of identities of users who have already joined the group and $\mathsf{N}_{\mathsf{max}}$ denote the maximum number of users the system supports. Upon receiving the identity $\mathsf{Username}$, if $\mathsf{Username} \in \mathsf{ID}$ or $|\mathsf{ID}| \geq \mathsf{N}_{\mathsf{max}}$, the algorithm outputs $\perp$. Otherwise, i.e. if $\mathsf{Username} \notin \mathsf{ID}$ and $|\mathsf{ID}| < \mathsf{N}_{\mathsf{max}}$, the manager $\mathcal{M}$ selects the smallest unassigned *identifier* id with $1 \leq \mathsf{id} \leq \mathsf{N}_{\mathsf{max}}$ and generates a corresponding *secret value* c_{id} and sends $(\mathsf{id}, c_{\mathsf{id}})$ to the user. Then, $\mathcal{M}$ stores(i) $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$ in a *private list* $\mathsf{PL}_{\mathcal{M}}$, where

Active indicates that the user is valid, and (ii) the user's identity $\mathsf{Username}$ as the id-th element in the list ID, i.e. $\mathsf{ID}[\mathsf{id}] = \mathsf{Username}$. Thus knowing the identifier id allows the manager to retrieve the identity Username and vice versa. Two lists $\mathsf{PL}_{\mathcal{M}}$ and ID are both initially empty and $\mathsf{PL}_{\mathcal{M}}$ is private. When a user with identifier id needs new keys, the user proves their identity by securely transmitting $(\mathsf{id}, c_{\mathsf{id}})$ to $\mathcal{M}$ through the secure channel. If $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active}) \in \mathsf{PL}_{\mathcal{M}}$, the manager $\mathcal{M}$ sends new keys to the user through the secure channel.

- $\sigma \leftarrow \mathsf{FDGS.Sig}(m, \mathbf{gsk}^{\mathsf{id}})$: This algorithm is run by a user with identifier id. It takes a message m and a key $\mathbf{gsk}^{\mathsf{id}}$ and generates a group signature σ.
- $0/1 \leftarrow \mathsf{FDGS.Vf}(m, \sigma, \mathsf{FDGS.PubPr})$: This is a deterministic verification algorithm run by a verifier. It takes as input a message m, a group signature σ, and the public parameters FDGS.PubPr, and outputs 1 if σ is a valid group signature on m with respect to FDGS.PubPr, and 0 otherwise.
- $\mathsf{id}/\perp \leftarrow \mathsf{FDGS.Op}(\sigma, \mathsf{FDGS.SK})$: This deterministic opening algorithm is run by $\mathcal{M}$. It takes a valid group signature σ and the manager's secret key FDGS.SK, and outputs either the identifier id of the user who generated σ, or $\perp$ if the signature cannot be attributed to any specific user. With id, the manager can retrieve the identity Username of the user by looking up the id-th entry in the list ID, where Username is stored.
- $(\mathsf{PL}_{\mathcal{M}}, \mathsf{FDGS.PubPr}) \leftarrow \mathsf{FDGS.Rev}(\mathsf{FDGS.SK}, \mathsf{PL}_{\mathcal{M}}, \mathsf{FDGS.PubPr}, \mathsf{R})$: The manager $\mathcal{M}$ runs this algorithm to revoke users with identifiers in a set R and invalidate their signatures. It takes as input a set R, the manager's secret key FDGS.SK, the private list $\mathsf{PL}_{\mathcal{M}}$, and the public parameters FDGS.PubPr, and updates both the public parameters FDGS.PubPr and the private list $\mathsf{PL}_{\mathcal{M}}$. Specifically, for each $\mathsf{id} \in \mathsf{R}$, the manager replaces $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$ with $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Revoked})$ in $\mathsf{PL}_{\mathcal{M}}$, preventing these users from requesting further keys.

3.1 Fully Dynamic Group Signature: Security Model

A fully dynamic group signature must satisfy four properties: *Correctness, Unforgeability, Anonymity*, and *Traceability*. Each of these properties is defined using the game, where each game is defined between a PPT adversary denoted by $\mathcal{A}$ and a *Challenger*. Each of these properties is quantified by the success probability of $\mathcal{A}$ in its game. In these games, the adversary has access to various oracles, each specifying their capabilities, which can be invoked a polynomially many times during the game.

We first describe the oracles in Table 3, and then in Definition 1 formally define the security properties of the signature scheme. The corresponding security experiments are given in Table 4. In the following, the notations $\mathcal{H}$, $\mathcal{C}$, and $\mathcal{R}$ denote the set of honest, corrupted, and revoked users, respectively. N, $\mathsf{N}_{\mathsf{max}}$, ID, and $\mathsf{PL}_{\mathcal{M}}$ denote the number of users who have already joined the group, the max number of users that the group supports, the list of identities of users (i.e. Usernames of the joined users) and the private list of manager. Also, CL and SL indicate the challenging list and the signing list.

- $\mathsf{AddHU}(\mathsf{Username})$: This oracle simulates the join protocol FDGS.Join between a user with identity Username and the manager $\mathcal{M}$. If $\mathsf{Username} \notin \mathsf{ID}$ and $N < \mathsf{N}_{\mathsf{max}}$, this oracle adds this user to the group as an honest user, i.e. adds $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$ to the private list $\mathsf{PL}_{\mathcal{M}}$ and stores the user's identity Username as the id-th element in the list ID, where id is the smallest unassigned identifier in $[1, \mathsf{N}_{\mathsf{max}}]$. Finally, it returns id to the adversary $\mathcal{A}$. $\mathcal{A}$ does not learn c_{id}.
- $\mathsf{AddCU}(\mathsf{id})$: This oracle allows the adversary $\mathcal{A}$ to corrupt an honest user with identifier id. It adds id to $\mathcal{C}$ and removes id from $\mathcal{H}$ and finally returns the secret value c_{id} to $\mathcal{A}$, enabling $\mathcal{A}$ to communicate with the manger $\mathcal{M}$ and receive this user's keys from $\mathcal{M}$.
- $\mathsf{Revoke}(\mathsf{R})$: This oracle allows the adversary $\mathcal{A}$ to revoke the set R of users by updating the private list $\mathsf{PL}_{\mathcal{M}}$ and the public parameters FDGS.PubPr. Specifically, for any $\mathsf{id} \in \mathsf{R}$, $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$ will be replaced with $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Revoked})$ in $\mathsf{PL}_{\mathcal{M}}$, preventing these ids from requesting further keys. Also, it invalidates their previously generated signatures by updating FDGS.PubPr.
- $\mathsf{Ch}_b(\mathsf{id}_0, \mathsf{id}_1, m)$: This oracle takes the identifiers id_0 and id_1 of two honest users along with a message m, then randomly selects a bit b and outputs the signature σ_b on m by an unused secret key of the user id_b. It keeps (m, σ_b) in the *challenge list* CL to prevent $\mathcal{A}$ from calling the *opening* oracle $\mathsf{Open}(\cdot,\cdot)$ on (m, σ_b) in the Anonymity experiment.
- $\mathsf{Open}(m, \sigma)$: This oracle takes as input a pair (m, σ). If σ is a valid signature on m and $(m, \sigma) \notin \mathsf{CL}$, it returns $\mathsf{FDGS.Op}(\sigma, \mathsf{FDGS.SK})$, which is either the identifier id of the user who produced the signature σ, or $\perp$ if σ cannot be attributed to any user.
- $\mathsf{SignHU}(\mathsf{id}, m)$: This oracle takes as input an identifier id and a message m. If this user is honest, it returns $\sigma \leftarrow \mathsf{FDGS.Sig}(m, \mathbf{gsk}^{\mathsf{id}})$, where $\mathbf{gsk}^{\mathsf{id}}$ is a secret key of this user. The oracle also adds (m, σ) to the signing list SL to prevent $\mathcal{A}$ from using (m, σ) in the Unforgeability experiment.

The Correctness property ensures that even if the adversary $\mathcal{A}$ corrupts a set of users (possibly all except one), the honest user can successfully enroll and create valid signatures that can be traced back to them. The Unforgeability property ensures that the adversary cannot produce a valid signature that can be falsely attributed to an honest

Table 3 Oracles

```
AddHU(Username)
1 : if Username $\in$ ID $\vee N \geq$ N$_{\max}$, then **return** $\perp$;
2 : ID = ID $\cup$ {Username}; $N = N+1$;
3 : (id, $c_{\mathsf{id}}$, Active) $\leftarrow$ FDGS.Join(Username);
4 : PL$_{\mathcal{M}}$ = PL$_{\mathcal{M}} \cup$ (id, $c_{\mathsf{id}}$, Active);
5 : $\mathcal{H} = \mathcal{H} \cup$ {id};
6 : **return** id;

AddCU(id)
1 : if id $\notin \mathcal{H}$, then **return** $\perp$;
2 : $\mathcal{C} = \mathcal{C} \cup$ {id}; $\mathcal{H} = \mathcal{H} \setminus$ {id};
3 : Retrieve (id, $c_{\mathsf{id}}$, Active) from the list PL$_{\mathcal{M}}$;
4 : **return** $c_{\mathsf{id}}$;

SignHU(id, $m$)
1 : if id $\notin \mathcal{H}$, then **return** $\perp$;
2 : Selects a key gsk$^{\mathsf{id}}$ of id;
3 : $\sigma \leftarrow$ FDGS.Sig($m$, gsk$^{\mathsf{id}}$);
4 : SL = SL $\cup$ {($m$, $\sigma$)}; **return** $\sigma$;

Open($m$, $\sigma$)
1 : if ($m$, $\sigma$) $\in$ CL $\vee$ FDGS.Vf($m$, $\sigma$, FDGS.PubPr) = 0, then **return** $\perp$;
2 : **return** FDGS.Op($\sigma$, FDGS.SK);

Revoke(R)
1 : if R $\not\subset \{1, \cdots, N\}$, then **return** $\perp$;
2 : R$'$ = R $\setminus \mathcal{R}$; $\mathcal{R}$ = R$' \cup \mathcal{R}$;
3 : (PL$_{\mathcal{M}}$, FDGS.PubPr) $\leftarrow$ FDGS.Rev(FDGS.SK, PL$_{\mathcal{M}}$, FDGS.PubPr, R$'$);
4 : **return** FDGS.PubPr;

Ch$_b$(id$_0$, id$_1$, $m$)
1 : if {id$_0$, id$_1$} $\not\subset \mathcal{H}$, then **return** $\perp$;
2 : Selects a key gsk$^{\mathsf{id}_b}$ of id$_b$;
3 : $\sigma_b \leftarrow$ FDGS.Sig($m$, gsk$^{\mathsf{id}_b}$);
4 : CL = {($m$, $\sigma_b$)};
5 : **return** $\sigma_b$;
```

user, even if the adversary corrupts the rest of the users. Anonymity is defined through an indistinguishability experiment between the adversary and the challenger. In this experiment, the adversary knows the secret keys of all users except two. The Anonymity property requires that the adversary has only a negligible advantage in distinguishing a signature generated by randomly selecting one of the two identifiers and using the corresponding private key to sign the message. Finally, Traceability ensures that all valid signatures that pass verification can be linked to a user.

Definition 1 For any security parameter $\lambda \in \mathbb{N}$ and for any PPT adversary $\mathcal{A}$, we say that an $\mathcal{FDGS}$ provides:

1. **Correctness** if there exists a negligible function ε_1 such that
$$\mathbf{Adv}^{\mathrm{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathrm{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1\right] \leq \varepsilon_1(\lambda),$$
2. **Unforgeability** if there exists a negligible function ε_2 such that
$$\mathbf{Adv}^{\mathrm{Unforg}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathrm{Unforg}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1\right] \leq \varepsilon_2(\lambda),$$
3. **Anonymity** if there exists a negligible function ε_3 such that
$$\mathbf{Adv}^{\mathrm{Anon}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathbf{Exp}^{Anon-0}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathbf{Exp}^{Anon-1}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1\right]\right| \leq \varepsilon_3(\lambda),$$
4. **Traceability** if there exists a negligible function ε_4 such that
$$\mathbf{Adv}^{\mathrm{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = \Pr\left[\mathbf{Exp}^{\mathrm{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1\right] \leq \varepsilon_4(\lambda),$$

where $\mathbf{Exp}^{\mathrm{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$, $\mathbf{Exp}^{\mathrm{Unforg}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$, $\mathbf{Exp}^{\mathrm{Anon-b}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$ and $\mathbf{Exp}^{\mathrm{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$ are defined in Table 4.

Table 4 Experiments

```
$\mathbf{Exp}^{\mathrm{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$
1 : (FDGS.SK, FDGS.PubPr) $\leftarrow$ FDGS.KG($1^{\lambda}$, FDGS.SetPr);
2 : $N = 0$; ID = $\emptyset$; id = $\perp$; $\mathcal{H} = \emptyset$; $\mathcal{C} = \emptyset$; $\mathcal{R} = \emptyset$;
3 : $m$, id $\leftarrow \mathcal{A}^{\mathsf{AddHU,AddCU,Revoke}}$(FDGS.PubPr);
4 : if id $\notin \mathcal{H}$, then **return** 0;
5 : $\sigma \leftarrow$ FDGS.Sig($m$, gsk$^{\mathsf{id}}$);
6 : if FDGS.Vf($m$, $\sigma$, FDGS.PubPr) = 0, then **return** 1;
7 : if FDGS.Op($\sigma$, FDGS.SK) $\neq$ id, then **return** 1;
8 : **return** 0;
```

$\mathbf{Exp}^{\mathrm{Unforg}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

```
1: $(\mathsf{FDGS.SK}, \mathsf{FDGS.PubPr}) \leftarrow \mathsf{FDGS.KG}(1^\lambda, \mathsf{FDGS.SetPr})$;
2: $N = 0$; $\mathsf{ID} = \emptyset$; $\mathcal{H} = \emptyset$; $\mathcal{C} = \emptyset$; $\mathcal{R} = \emptyset$; $\mathsf{SL} = \emptyset$;
3: $(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{AddCU},\mathsf{SignHU},\mathsf{Revoke}}(\mathsf{FDGS.PubPr})$;
4: if $(m, \sigma) \in \mathsf{SL} \vee \mathsf{FDGS.Vf}(m, \sigma, \mathsf{FDGS.PubPr}) = 0$, then **return** 0;
5: if $\mathsf{FDGS.Op}(\sigma, \mathsf{FDGS.SK}) \notin \mathcal{H}$, then **return** 0;
6: **return** 1;
```

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

```
1: $(\mathsf{FDGS.SK}, \mathsf{FDGS.PubPr}) \leftarrow \mathsf{FDGS.KG}(1^\lambda, \mathsf{FDGS.SetPr})$;
2: $N = 0$; $\mathsf{ID} = \emptyset$; $\mathcal{H} = \emptyset$; $\mathcal{C} = \emptyset$; $\mathcal{R} = \emptyset$; $\mathsf{CL} = \emptyset$;
3: $b' \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{AddCU},\mathsf{SignHU},\mathsf{Revoke},\mathsf{Open},\mathsf{Ch}_b}(\mathsf{FDGS.PubPr})$;
4: **return** $b'$;
```

$\mathbf{Exp}^{\mathrm{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

```
1: $(\mathsf{FDGS.SK}, \mathsf{FDGS.PubPr}) \leftarrow \mathsf{FDGS.KG}(1^\lambda, \mathsf{FDGS.SetPr})$;
2: $N = 0$; $\mathsf{ID} = \emptyset$; $\mathcal{H} = \emptyset$; $\mathcal{C} = \emptyset$; $\mathcal{R} = \emptyset$;
3: $(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{AddCU},\mathsf{SignHU},\mathsf{Revoke}}(\mathsf{FDGS.PubPr})$;
4: if $\mathsf{FDGS.Vf}(m, \sigma, \mathsf{FDGS.PubPr}) = 0$, then **return** 0;
5: if $\mathsf{FDGS.Op}(\sigma, \mathsf{FDGS.SK}) = \perp$, then **return** 1;
6: **return** 0;
```

3.2 An Overview of DGM

DGM is a fully dynamic group signature scheme based on symmetric primitives that uses a collision-resistant hash function H, an $\mathcal{OTS}$ scheme, a symmetric puncturable encryption scheme $\mathcal{SPE}$, and a symmetric key encryption scheme $\mathcal{SE}$. DGM employs two types of Merkle trees: One Initial Merkle tree (IMT) and multiple Signing Merkle trees (SMT). This IMT/SMT structure allows the manager to gradually construct the tree, avoiding the unacceptable computational cost of building the entire tree during the setup phase. In the following, we review DGM and mention its main shortcomings. See [11] for further details.

In DGM, the IMT is of height 20 and its leaf nodes are randomly chosen binary strings. The root of the IMT is the group public key $\mathsf{DGM.gpk}$. SMTs are variable-height Merkle trees and their leaf nodes correspond to the OTSs that are used to sign messages by users. Each SMT is connected to only one internal node of the IMT. While DGM allows multiple SMTs per fallback node, this is not formalized or included in the security proof. The height of an SMT is equal to $20 - h^*$, where h^* is the height of the corresponding internal node in the IMT. When a user requires B signing keys, they send a request to the manager, who randomly selects B internal nodes from the IMT and allocates the next available B OTSs from the B SMTs linked to those internal nodes. If an SMT has no available OTS, the manager generates a new SMT, links it to the corresponding internal node, and assigns an OTS from the newly created SMT.

SMT Generation. SMTs are generated gradually and linked randomly to the internal nodes of IMT and their leaf nodes will be distributed among the users. Let the height of the t-th SMT be h'. The DGM manager generates $z = 2^{h'}$ OTS key pairs $(\mathsf{OTS.sk}_{t,l}, \mathsf{OTS.pk}_{t,l})$ for $0 \leq l < z$, and *shuffles* the leaf nodes of this SMT, indexed by $\{(t,l)\}_{l=0}^{z-1}$, using the symmetric encryption scheme $\mathcal{SE}$. More precisely, the leaf nodes are constructed as $\mathsf{H}(\mathsf{OTS.pk}_{t,l} \parallel \mathsf{DGM.pos}_{t,l})$, where $\mathsf{DGM.pos}_{t,l} \leftarrow \mathsf{SE.Enc}(\mathsf{msk}, t \parallel l)$, with msk being the manager's secret key. These nodes are then sorted in increasing order based on their $\mathsf{DGM.pos}_{t,l}$ values for $0 \leq l < z$. We note that after sorting the leaf nodes, the l-th leaf node will be located at index l'.

Let r_t be the root of the t-th SMT, which is connected to an internal node of the IMT, denoted by Fn_i (called *fallback node*), using a *fallback key* $Fk_t \leftarrow \mathsf{SE.Dec}(r_t, Fn_i)$ (See Figure 3). The signature of a message m by $\mathsf{OTS.sk}_{t,l}$ is

$$\sigma_{DGM} = ((i, l'), \mathsf{OTS}.\sigma_{t,l}, \mathsf{OTS.pk}_{t,l}, \mathsf{DGM.pos}_{t,l}, Fk_t, \mathsf{DGM.path}_{i,l'}),$$

where i is the index of fallback node Fn_i, l' is the index of the leaf node of the OTS key pair $(\mathsf{OTS.sk}_{t,l}, \mathsf{OTS.pk}_{t,l})$ after sorting the leaf nodes, $\mathsf{OTS}.\sigma_{t,l} \leftarrow \mathsf{OTS.Sig}(\mathsf{OTS.sk}_{t,l}, m)$, and $\mathsf{DGM.path}_{i,l'}$ is the authentication path from the leaf node with index (i, l') in SMT to the root of IMT. See figure 3 for the IMT/SMT structure of DMG.

DGM Limitations. There are two main shortcomings in DGM. First, during verification, the verifier must interact with the manager to check the validity of the fallback key Fk_t, requiring the manager to be always online. This creates a system bottleneck and a single point of failure. Second, the manager must maintain a list containing all the allocated $\mathsf{DGM.pos}_{t,l}$ to users for the purpose of opening signatures via list-based searches, and revoking users. Thus, the manager's storage grows linearly with the total number of signatures in the system and becomes unacceptably large when a large number of signatures is required. For instance, to accommodate $\mathsf{T}_{\mathsf{tot}} = 2^{64}$ OTSs, the manager requires $\sim 10^{8.7}$ Terabytes of storage (see [51] for details).

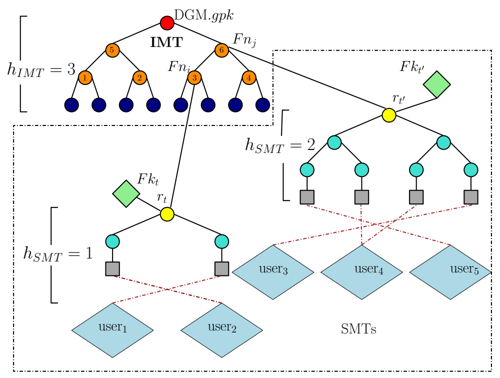

Fig. 3 IMT/SMT structure of DGM. The red node represents the group public key DGM.gpk, the orange nodes represent the fallback nodes $\{Fn_i\}_{i=1}^{6}$, the green nodes Fk_t and $Fk_{t'}$ represent the fallback keys, and the yellow nodes are the root of the SMTs. The fallback key Fk_t is computed as $Fk_t \leftarrow \mathsf{SE.Dec}(r_t, Fn_i)$, where r_t is the key.

4 DGMT: A Flexible Fully Dynamic Symmetric-key based GSS

DGMT is an improved version of DGM that removes DGM's notable shortcomings, such as the need for interactive verification and the unreasonable storage requirements proportional to $\mathsf{T_{tot}}$, the total number of supported signatures. In this section, we first present an overview of DGMT in Subsction 4.1, then explain the details of the DGMT's tree construction and its algorithms in Subsections 4.2 and 4.3, respectively. Finally, we discuss the typical parameters of DGMT in Subsection 4.4. A list of the main symbols are summarized in Table 5.

4.1 DGMT: An Overview

DGMT uses a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ that is randomly selected from a collision-resistant hash function family $\mathcal{HF}$, an EU-CMA one-time signature scheme $\mathcal{OTS} = (\mathsf{OTS.KG}, \mathsf{OTS.Sig}, \mathsf{OTS.Vf})$, a PRF $\mathsf{f} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, and two SPRPs $\mathsf{g}_1 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ and $\mathsf{g}_2 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$.

Tree structure. DGMT uses the IMT/SMT$^{\text{MT}}$ structure, which is similar to the IMT/SMT structure of DGM, with two key modifications outlined below. First, similar to DGM, the IMT is a Merkle tree generated over a set of randomly chosen leaf nodes and its root forms the group public key, denoted by DGMT.gpk. However, in DGMT all nodes of IMT, except the root node, are used as fallback nodes. Second, unlike DGM which uses SMTs with variable heights, DGMT employs SMT$^{\text{MT}}$s with the same height, where each SMT$^{\text{MT}}$ is a two-layer Merkle tree, and both layers have the same height h_S. The root node of an SMT$^{\text{MT}}$ is connected to a fallback node using a fallback key. A fallback node is used to attach γ number of SMT$^{\text{MT}}$s, each using a distinct fallback key, to the IMT. We use the numbering of IMT nodes and the multiplicity number of SMT$^{\text{MT}}$s that are attached to an IMT node to give a unique sequential index to each SMT$^{\text{MT}}$. Thus the j-th SMT$^{\text{MT}}$ that is connected to the i-fallback node Fn_i will be labeled SMT$^{\text{MT}}_{\text{i,j}}$ and will be associated with the fallback key $Fk_{i,j}$. Figure 4 illustrates the IMT/SMT$^{\text{MT}}$ structure of DGMT.

The top and bottom layers of the two-layer SMT$^{\text{MT}}$ are denoted by SMT$^{(1)}$ and SMT$^{(2)}$, respectively. In particular, the SMT$^{(1)}$ of the SMT$^{\text{MT}}_{\text{i,j}}$ is denoted by SMT$^{(1)}_{\text{i,j}}$ and the SMT$^{(2)}$ of the SMT$^{\text{MT}}_{\text{i,j}}$ linked to the k-th leaf node (from left) of SMT$^{(1)}_{\text{i,j}}$ is denoted by SMT$^{(2)}_{\text{i,j,k}}$. From now on, we let (i,j,k) be the index of the k-th leaf node of SMT$^{(1)}_{\text{i,j}}$, and (i,j,k,l) be the index of the l-th leaf node of SMT$^{(2)}_{\text{i,j,k}}$.

The manager uses the PRF f and a secret key $\mathsf{SMT_1.key}$ to generate OTS key pairs $(\mathsf{OTS.sk}_{i,j,k}, \mathsf{OTS.pk}_{i,j,k})$, for $0 \le k < 2^{h_s}$, and construct the SMT$^{(1)}_{\text{i,j}}$ as an MSS, as described in Subsection 2.3. Also, the manager uses the PRF f and a secret key $\mathsf{SMT_2.key}$ to generate the OTS key pairs $(\mathsf{OTS.sk}_{i,j,k,l}, \mathsf{OTS.pk}_{i,j,k,l})$, for $0 \le l < 2^{h_s}$, and construct SMT$^{(2)}_{\text{i,j,k}}$ for all i,j,k. However, for SMT$^{(2)}_{\text{i,j,k}}$s the manager first permutes the set $\{(i,j,k,l) \mid 0 \le l < 2^{h_s}\}$ using the SPRP g_1 (See Algorithm 2). Let the index (i,j,k,l') be the permuted position of the index (i,j,k,l). After permutation, the leaf node at index (i,j,k,l'), for $0 \le l' < 2^{h_s}$, is computed as $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \parallel \mathsf{DGMT.pos}_{i,j,k,l})$, where $\mathsf{DGMT.pos}_{i,j,k,l} \leftarrow \mathsf{g}_2(\mathsf{msk}, i \parallel j \parallel k \parallel l)$ and msk is the manager's master secret key of the SPRP g_2. Finally, SMT$^{(2)}_{\text{i,j,k}}$ is constructed as an MSS using these permuted leaf nodes. See Algorithm 3.

Setup phase. During the setup phase, the manager performs the following steps:(i) Generates the group secret keys DGMT.SK, (ii) Constructs the IMT to compute the group public key DGMT.gpk, (iii) Constructs all SMT$^{(1)}_{\text{i,j}}$, for all valid i and j, and uses the root of each SMT$^{(1)}_{\text{i,j}}$, denoted by $r_{i,j}$, to compute its corresponding fallback key $Fk_{i,j} \leftarrow \mathsf{g}_1(r_{i,j}, Fn_i)$, linking SMT$^{\text{MT}}_{\text{i,j}}$ to the internal node Fn_i of IMT, and iv) Publishes the group public parameter $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$, where $\mathcal{FK}$ is the list of all fallback keys and $\mathcal{RL}$ is the revocation list, initially empty. There are γ fallback keys for each IMT internal node, and $Fk_{i,j}$ is stored as the $(\gamma(i-1)+j)$-th element in the list $\mathcal{FK}$. See Subsection 4.2 specifically Algorithm 1.

Join and Request OTS key pairs. In DGMT, a user first joins the group and receives $(\mathsf{id}, c_{\mathsf{id}})$. When the user with identifier id needs OTS key pairs, they send $(\mathsf{id}, c_{\mathsf{id}})$ as a request for new keys to the manager. Upon receiving this request, if $(\mathsf{id}, c_{\mathsf{id}})$ is valid and the user has not been revoked (i.e. $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active}) \in \mathsf{PL}_{\mathcal{M}}$), the manager randomly selects B fallback nodes $\{Fn_i\}_{i=1}^{B}$ and allocates the first available

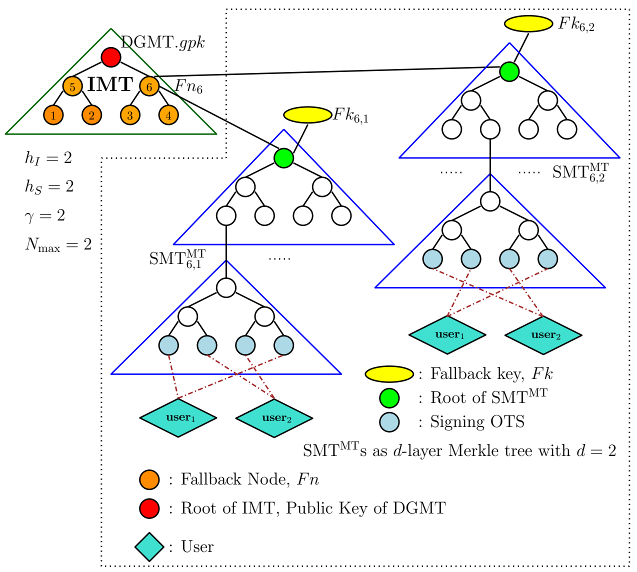

Fig. 4 IMT/SMT$^{\text{MT}}$ structure of DGMT. The red node represents the group public key DGMT.gpk, the orange nodes represent the fallback nodes $\{Fn_i\}_{i=1}^{6}$, the yellow nodes $Fk_{6,1}$ and $Fk_{6,2}$ represent the fallback keys, the green nodes are the root of the SMT$^{\text{MT}}$s, and the fallback key $Fk_{6,1} \leftarrow \mathsf{g}_1(r_{6,1}, Fn_6)$, where $r_{6,1}$ is the key

OTS key pair from the corresponding SMT$^{\text{MT}}$s. If no available OTS keys remain in a selected SMT$^{\text{MT}}$, the manager deterministically generates a new SMT$^{\text{MT}}$, ensuring the fallback keys match those computed during the setup phase. This request can be repeated when all OTS keys are exhausted to receive a new keys. This process is similar to the approach used in DGM. See Subsection 4.2.

Signing. The format of the DGMT signature σ_{DGMT} on a message m, generated using the OTS key pair $(\mathsf{OTS.sk}_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'})$ at index (i, j, k, l'), is as follows:

$$\begin{aligned}\sigma_{\text{DGMT}} = \big(&(i,j,k,l'), \mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k,l'}, m), \mathsf{OTS.pk}_{i,j,k,l'},\\ &\mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k}, r_{i,j,k}),\\ &\mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i\big). \qquad (1)\end{aligned}$$

$\mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k,l'}, m)$ is the only part of σ_{DGMT} that is computed by signer, while the remainder is provided by the manager during the OTS key pair request phase. Each signature has a unique and secret value $\mathsf{DGMT.pos}_{i,j,k,l}$ that is used for both tracing and revoking signatures. The authentication path from the leaf node indexed by (i, j, k, l') to DGMT.gpk consists of three parts:(i) the authentication path $A.path_{i,j,k,l'}$ from the leaf node at index (i, j, k, l') to the root $\text{SMT}^{(2)}_{\text{i,j,k}}$, denoted by $r_{i,j,k}$, (ii) the authentication path $A.path_{i,j,k}$ from the leaf node at index (i, j, k) to the root $\text{SMT}^{(1)}_{\text{i,j}}$, denoted by $r_{i,j}$, and (iii) the authentication path $A.path_i$ from Fn_i to DGMT.gpk. Moreover, $\mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k}, r_{i,j,k})$ is the signature on $r_{i,j,k}$ by the OTS key pair $(\mathsf{OTS.sk}_{i,j,k}, \mathsf{OTS.pk}_{i,j,k})$. See Algorithm 7.

Revoking a user. DGMT employs a SPRP g_2 to compute and add all assigned DGMT.poss of a misbehaving user to the revocation list $\mathcal{RL}$. In DGMT, each user is assigned a unique interval, allowing the manager to run this process efficiently without the need to store all users' DGMT.poss in advance, computing them only when necessary. See Algorithm 5 in Subsection 4.3. In contrast, in DGM, the manager uses a symmetric puncturable encryption scheme $\mathcal{SPE}$ to puncture all $\mathsf{DGM.pos}_{t,l}$ of a misbehaved user. Our revocation list based method requires significantly less storage and computation compared to $\mathcal{SPE}$-based revocation of DGM. See Section 6 for a detailed analysis.

Verification. Verifying a signature σ_{DGMT} on a message m involves the following four steps: (i) Ensure that $\mathsf{DGMT.pos}_{i,j,k,l} \notin \mathcal{RL}$, (ii) Verify $\mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k,l'}, m)$ and $\mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k}, r_{i,j,k})$ using the public keys $\mathsf{OTS.pk}_{i,j,k,l'}$ and $\mathsf{OTS.pk}_{i,j,k}$, (iii) Compute $Fn_i \leftarrow \mathsf{g}_1^{-1}(r_{i,j}, \mathcal{FK}[(\gamma(i-1)+j)])$, where $\mathcal{FK}[(\gamma(i-1)+j)]$ is the $(\gamma(i-1)+j)$-th element of the list $\mathcal{FK}$, and iv) Verify that the computed group public key from $\mathsf{DGMT.pos}_{i,j,k,l}$, the authentication path, and Fn_i equals to DGMT.gpk. See Algorithm 8 for details. Unlike DGM, DGMT's verification is non-interactive, and the fallback key is not included in the signature.

Opening. DGMT opens the signature σ_{DGMT} by computing $i \parallel j \parallel k \parallel l \leftarrow \mathsf{g}_2^{-1}(\mathsf{msk}, \mathsf{DGMT.pos}_{i,j,k,l})$. Indeed, the method of distributing leaf nodes among users and including $\mathsf{DGMT.pos}_{i,j,k,l}$ as part of the signature σ_{DGMT} allows the manager to trace each signature using the secret key msk. See Algorithm 10 for details. In contrast, in DGM, the manager must store $\mathsf{DGM.pos}_{t,l}$ in a list and search through the list to open each signature, a process that becomes prohibitively expensive for large $\mathsf{T}_{\mathsf{tot}}$.

4.2 Constructing DGMT Tree

In this subsection, we provide a detailed explanation of the DGMT tree framework and discuss how its design effectively removes the limitations of DGM.

4.2.1 Setup parameters

DGMT is designed to support $\mathsf{T}_{\mathsf{tot}}$ signatures and $\mathsf{N}_{\mathsf{max}}$ users. Based on these values, DGMT selects three parameters h_I, h_{SM}, and γ and defines its *setup parameters* as

$$\mathsf{DGMT.SetPr} = (h_I, h_{SM}, \gamma, \mathsf{T}_{\mathsf{tot}}, \mathsf{N}_{\mathsf{max}}),$$

where h_I, h_{SM}, and γ represent the height of the IMT, the height of the SMT$^{\text{MT}}$s, and the number of SMT$^{\text{MT}}$s attached

Table 5 List of Symbols

IMT	Initial Merkle Tree
r_{IMT}	The root of IMT
Fn_i	The i-th internal node of IMT (the i-th fallback node)
$\mathrm{SMT}^{\mathrm{MT}}$	Multilayer Signing Merkle Tree; We use 2-layer tree
$\mathrm{SMT}^{(1)}$	First (or Upper) layer of $\mathrm{SMT}^{\mathrm{MT}}$
$\mathrm{SMT}^{(2)}$	Second (or Lower) layer of $\mathrm{SMT}^{\mathrm{MT}}$
$\mathrm{SMT}^{\mathrm{MT}}_{\mathrm{i,j}}$	The j-th $\mathrm{SMT}^{\mathrm{MT}}$ that is attached to Fn_i
$\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$	$\mathrm{SMT}^{(1)}$ of $\mathrm{SMT}^{\mathrm{MT}}_{\mathrm{i,j}}$
$\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$	$\mathrm{SMT}^{(2)}$ attached to the k-th leaf node (from left) of $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$
(i,j,k)	The index of the k-th leaf node of $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$
(i,j,k,l)	The index of the l-th-leaf node of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$
h_I, h_{SM}, h_S	Heights of IMT, $\mathrm{SMT}^{\mathrm{MT}}$, and $\mathrm{SMT}^{(1)}$ ($\mathrm{SMT}^{(2)}$), respectively
α	The number of leaves of $\mathrm{SMT}^{(1)}$ and $\mathrm{SMT}^{(2)}$; $\alpha = 2^{h_S}$
$\mathsf{T}_{\mathsf{tot}}$	The total number of signatures that DGMT supports
$\mathsf{N}_{\mathsf{max}}$	The maximum member of users that DGMT supports
β	The number of keys allocated to a user in each $\mathrm{SMT}^{(2)}$
msk	The manager's master secret key of the SPRP g_2
$\mathsf{IMT.key}$	The key of the PRF f for generating the leaf nodes of the IMT
$\mathsf{SMT_1.key}$	The key of the PRF f for generating the leaf nodes of $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$
$\mathsf{SMT_2.key}$	The key of the PRF f for generating the leaf nodes of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$
$\mathsf{shuffle.key}$	The secret key of the SPRP g_1 for shuffling the leaves of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$
$(\mathsf{OTS.sk}_{i,j,k}, \mathsf{OTS.pk}_{i,j,k})$	OTS key pair corresponding to the leaf node at index (i,j,k)
$(\mathsf{OTS.sk}_{i,j,k,l}, \mathsf{OTS.pk}_{i,j,k,l})$	OTS key pair corresponding to the leaf node at index (i,j,k,l)
$r_{i,j}$	The root of the $\mathrm{SMT}^{\mathrm{MT}}_{\mathrm{i,j}}$ (or $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$)
$r_{i,j,k}$	The root of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$
γ	The number of $\mathrm{SMT}^{\mathrm{MT}}$s linked to each internal node of IMT
$\mathcal{FK}$	The list of fallback keys
$Fk_{i,j}$	The fallback key that links $\mathrm{SMT}^{\mathrm{MT}}_{\mathrm{i,j}}$ to the internal node Fn_i. $Fk_{i,j}$ is stored as the $((i-1)\gamma + j)$-th elemenet of the list $\mathcal{FK}$.
$\mathcal{RL}$	The revocation list
$\mathcal{I}_t$	The interval $[\beta(t-1), t\beta - 1]$ for index t
$\mathsf{DGMT.SetPr}$	The setup parameters of DGMT; $(h_I, h_{SM}, \gamma, \mathsf{T}_{\mathsf{tot}}, \mathsf{N}_{\mathsf{max}})$
$\mathsf{DGMT.SK}$	The secret keys $(\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$
$\mathsf{DGMT.gpk}$	The group public key, which is the root of IMT, i.e. r_{IMT}
$\mathsf{DGMT.PubPr}$	The group public parameters $(\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$
$\mathcal{L}_{i,j,k}$	The shuffled index list of leaves of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$
$\sigma_{i,j,k}$	The signature $\mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k}, r_{i,j,k})$
$\sigma_{i,j,k,l}$	The signature $\mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k,l}, m)$
$\mathsf{DGMT.pos}_{i,j,k,l}$	$\mathsf{g}_2(\mathsf{msk}, i \parallel j \parallel k \parallel l)$
$A.path_i$	Authentication path from Fn_i to r_{IMT}
$A.path_{i,j,k}$	Authentication path from leaf node (i,j,k) to r_{ij}
$A.path_{i,j,k,l'}$	Authentication path from leaf node (i,j,k,l') to $r_{i,j,k}$

to each fallback node, respectively. These parameters provide flexibility to the system, allowing it to efficiently generate at least $\mathsf{T}_{\mathsf{tot}}$ OTS key pairs and accommodate $\mathsf{N}_{\mathsf{max}}$ users. DGMT also uses two public lists:(i) A list $\mathcal{FK}$ of the fallback keys, and (ii) A revocation list $\mathcal{RL}$ of revoked signatures. The following relations hold in the IMT/SMT$^{\mathrm{MT}}$ structure of DGMT.

1. Each SMT$^{\mathrm{MT}}$ is a two-layer Merkle tree with layers of equal height h_S, so $h_{SM} = 2h_S$.
2. Given the parameters h_I, h_{SM}, and γ in DGMT, the total number of OTS key pairs is $\mathsf{T}_{\mathsf{tot}} = \gamma(2^{h_I+1}-2)2^{h_{SM}}$.
3. The number of fallback keys in DGMT for the given setup parameters $\mathsf{DGMT.SetPr}$ is $|\mathcal{FK}| = \gamma(2^{h_I+1}-2) = \mathsf{T}_{\mathsf{tot}}/2^{h_{SM}}$.
4. Each user must receive at least two keys from each SMT$^{(2)}_{\mathrm{i,j,k}}$, for every (i,j,k), thus h_S must satisfy $\mathsf{N}_{\mathsf{max}} \le \alpha/2$, where $\alpha = 2^{h_S}$ (See the Anonymity proof in Section 5).
5. For each tuple (i,j,k) and (i,j,k,l), we have $1 \le i < 2^{h_I+1}-1$, $1 \le j \le \gamma$, and $0 \le k,l \le \alpha-1$, where $\alpha = 2^{h_S}$.

Based on the second and third relations mentioned above, for a given $\mathsf{T}_{\mathsf{tot}}$ the number of fallback keys is $|\mathcal{FK}| = \gamma(2^{h_I+1}-2) = \mathsf{T}_{\mathsf{tot}}/2^{h_{SM}}$. Therefore, if the manager selects h_{SM} so that $\mathsf{T}_{\mathsf{tot}}/2^{h_{SM}}$ is sufficiently small, the manager can effectively generate and publish all the fallback keys at the setup phase. This allows the verifier to check the validity of the fallback keys without interacting with the manager, making DGMT a non-interactive group signature scheme and addressing the first shortcoming of DGM.

Remark 2 For a given $\mathsf{T}_{\mathsf{tot}}$, the parameters h_I, h_{SM}, and γ are chosen such that $\mathsf{T}_{\mathsf{tot}} = \gamma(2^{h_I+1}-2)2^{h_{SM}}$, ensuring $|\mathcal{FK}| = \mathsf{T}_{\mathsf{tot}}/2^{h_{SM}}$ is sufficiently small. This creates a trade-off between γ and h_I: a smaller γ results in a larger IMT. Since IMT nodes are stored in memory, a larger h_I increases memory costs. Therefore, γ must be large enough to have a reasonable h_I. The terms "sufficiently small" and "large enough" depend on the available system and communication cost.

4.2.2 Tree Construction and allocation of intervals to users

Constructing the DGMT tree is a two-step process.

1. In the setup phase, the manager constructs the IMT along with all SMT$^{(1)}_{\mathrm{i,j}}$s, for every i and j, to compute and publish $\mathsf{DGMT.gpk}$ and

   $$\mathcal{FK} = [Fk_{i,j} \leftarrow \mathsf{g}_1(r_{i,j}, Fn_i) \mid 1 \le i < 2^{h_I+1}-1,\ 1 \le j \le \gamma],$$

   where $r_{i,j}$ is the root of SMT$^{\mathrm{MT}}_{\mathrm{i,j}}$ and Fn_i is the i-th fallback node. See Algorihtm 1.
2. During the join phase when a user requests keys (or during the sign phase when a user requests new OTS key pairs) and no available key pairs remain in some current SMT$^{(2)}$s for this user, the manager must construct new SMT$^{(2)}$s. The manager then randomly selects new OTS key pairs from these newly constructed trees and assigns them to the user.

To explain the different steps of constructing DGMT's tree, we assume the manager has generated the secret key $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$, where each of them is a λ-bit random strings (See Algorithm 4 for details). Also, let $\mathsf{DGMT.SetPr} = (h_I, h_{SM}, \gamma, \mathsf{T}_{\mathsf{tot}}, \mathsf{N}_{\mathsf{max}})$ be the setup parameters. Thus, the number of leaf nodes in SMT$^{(1)}$s and SMT$^{(2)}$s is $\alpha = 2^{h_S}$ and $h_{SM} = 2h_S$.

Algorithm 1 $\mathsf{DGMT.PubPrCons}$

Input: The security parameter λ, $\mathsf{DGMT.SetPr} = (h_I, h_{SM}, \gamma, \mathsf{T}_{\mathsf{tot}}, \mathsf{N}_{\mathsf{max}})$, and $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$.
Output: $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$.

```
1: $\mathcal{RL} = [\,]$;  $\mathcal{FK} = [\,]$;
2: /* Construction of IMT */
3: for $1 \le i \le 2^{h_I}$ do;
4:     Compute $\mathsf{f}(\mathsf{IMT.key}, i)$;
5: end for
6: Construct a Merkle tree using $\{\mathsf{f}(\mathsf{IMT.key}, i)\}_{i=1}^{2^{h_I}}$, called IMT; Let its root be $r_{\mathrm{IMT}}$;
7: $\mathsf{DGMT.gpk} = r_{\mathrm{IMT}}$;
8: /* Construction of the Fallback keys */
9: for $1 \le i < 2^{(h_I+1)} - 1$ do
10:     for $1 \le j \le \gamma$ do
11:         for $0 \le k \le \alpha - 1$ do
12:             $\mathsf{OTS.sk}_{i,j,k} \leftarrow \mathsf{f}(\mathsf{SMT_1.key}, i \parallel j \parallel k)$;
13:             Compute $\mathsf{OTS.pk}_{i,j,k}$ from $\mathsf{OTS.sk}_{i,j,k}$;
14:         end for
15:         Construct a Merkle tree using $\{\mathsf{OTS.pk}_{i,j,k}\}_{k=0}^{\alpha-1}$, called SMT$^{(1)}_{\mathrm{i,j}}$; Let the root of SMT$^{(1)}_{\mathrm{i,j}}$ be $r_{i,j}$;
16:         $Fk_{i,j} \leftarrow \mathsf{g}_1(r_{i,j}, Fn_i)$;
17:         Append $Fk_{i,j}$ to the list $\mathcal{FK}$;
18:     end for
19: end for
20: $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$;
21: return $\mathsf{DGMT.PubPr}$;
```

1. **Constructing the IMT and SMT$^{(1)}$s and publishing $\mathsf{DGMT.PubPr}$:** The manager runs this process to publish $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$. This process is detailed in Algorithm 1, where the $\mathsf{DGMT.gpk}$ is computed in lines 3–7 and $\mathcal{FK}$ is computed in lines 9–19. The fallback key for SMT$^{\mathrm{MT}}_{\mathrm{i,j}}$ is $Fk_{i,j} \leftarrow \mathsf{g}_1(r_{i,j}, Fn_i)$, where $r_{i,j}$ serves as the key for g_1 (line 16). $\mathcal{FK}$ is a public list, and its $(\gamma(i-1)+j)$-th element is $Fk_{i,j}$, where $1 \le i < 2^{h_I+1}-1$ and $1 \le j \le \gamma$. Also, $\mathcal{RL}$ is initially

empty and will be updated by the Algorithm 9 whenever the manager revokes a misbehaving user.

2. **Constructing $\mathsf{SMT}^{(2)}$s and allocating their leaf nodes to users:** The manager constructs $\mathsf{SMT}^{(2)}$s when they want to randomly allocate the leaf nodes to users. Each user $\mathsf{id} \in [1, \mathsf{N}_{\max}]$ is assigned a unique interval

$$\mathcal{I}_{\mathsf{id}} = [\beta(\mathsf{id}-1), \beta\mathsf{id}-1] = \{\beta(\mathsf{id}-1), \cdots, \beta\mathsf{id}-1\},$$

for the following purposes:

(a) *Efficient Random Allocation.* The manager shuffles the leaf nodes of $\mathsf{SMT}^{(2)}$ using Algorithm 2, which takes an index (i,j,k) and $\mathsf{shuffle.key}$, and outputs the shuffled index list of leaves of $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$ as

$$\mathcal{L}_{i,j,k} = [l'_0, \ldots, l'_{\beta-1}, \ldots, l'_{\beta(\mathsf{N}_{\max}-1)}, \ldots, l'_{\beta\mathsf{N}_{\max}-1}].$$

For each $1 \le \mathsf{id} \le \mathsf{N}_{\max}$, sublist $[l'_{\beta(\mathsf{id}-1)}, \cdots, l'_{\beta\mathsf{id}-1}]$ represents the β leaf nodes of $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$ allocated to user id, which the manager uses to assign the corresponding OTS key pairs to user id.

(b) *Efficient Opening and Revocation.* Each signature σ_{DGMT} contains a unique $\mathsf{DGMT.pos}_{i,j,k,l} \leftarrow \mathsf{g}(\mathsf{msk}, i \,\|\, j \,\|\, k \,\|\, l)$. To open a signature, the manager computes $i \,\|\, j \,\|\, k \,\|\, l \leftarrow \mathsf{g}_2^{-1}(\mathsf{msk}, \mathsf{DGMT.pos}_{i,j,k,l})$ and identifies the identifier id as the signer if $l \in \mathcal{I}_{\mathsf{id}}$ (or equivalently announce $\mathsf{id} = \lceil \frac{l+0.5}{\beta} \rceil$ as the signer). This approach results in an efficient signature opening algorithm. Additionally, by using these intervals, the manager only needs to compute and add some specific $\mathsf{DGMT.pos}_{i,j,k,l}$s to the revocation list $\mathcal{RL}$, ensuring efficient signature revocation. See Algorithms 9 and 10.

Remark 3 As discussed in Section 3.2, DGM's second limitation is its inefficient management of storage, which scales linearly with $\mathsf{T}_{\mathsf{tot}}$. Indeed, this inefficiency arises from its simple random allocation of leaf nodes, requiring the manager to store all $\mathsf{DGM.pos}_{t,l}$ values, for each t and l, to enable signature opening and revocation. This approach demands an impractical amount of storage, requiring approximately $10^{8.7}$ terabytes to support $\mathsf{T}_{\mathsf{tot}} = 2^{64}$ OTSs [51]. In contrast, our proposed interval-based method overcomes this limitation and reduces storage requirements to scale with the number of intervals (or equivalently $\mathsf{N}_{\max}$), rather than $\mathsf{T}_{\mathsf{tot}}$.

To construct $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$, the manager first computes $\mathcal{L}_{i,j,k}$. The l-th element in this list indicates the position of the leaf node (i,j,k,l) after permuting the leaf nodes. Then, the manager computes the OTS key pairs $(\mathsf{OTS.sk}_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'})$ and $\mathsf{DGMT.pos}_{i,j,k,l}$, for all $0 \le l, l' \le \alpha - 1$. The l'-th leaf node of $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$ is $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \,\|\, \mathsf{DGMT.pos}_{i,j,k,l})$. After computing all the leaf nodes of this tree, the manager constructs $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$ and signs its root, $r_{i,j,k}$, by $\mathsf{OTS.sk}_{i,j,k}$ to attach $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$ to $\mathsf{SMT}^{(1)}_{\mathsf{i,j}}$. See Algorithm 3.

Algorithm 2 DGMT.Shuffle

Input: The secret key $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$ and the index (i,j,k) (the index of the k-th leaf node of $\mathsf{SMT}^{(1)}_{\mathsf{i,j}}$).

Output: $\mathcal{L}_{i,j,k}$, the shuffled index list of leaves of $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$

```
1: $\mathcal{L}_{i,j,k} = [\,]$;
2: $\mathcal{L}_1 = [\mathsf{g}_1(\mathsf{shuffle.key}, i \,\|\, j \,\|\, k \,\|\, l) \mid 0 \le l \le \alpha - 1]$;
3: $\mathcal{L}_2 \leftarrow Sort(\mathcal{L}_1)$;        ▷ Sorting $\mathcal{L}_1$ in ascending integer order
4: for $0 \le l \le \alpha - 1$ do
5:     Append the position of $\mathcal{L}_1[l]$ in the list $\mathcal{L}_2$ to the list $\mathcal{L}_{i,j,k}$;
6: end for;
7: return $\mathcal{L}_{i,j,k}$;
```

Algorithm 3 DGMT.SMTTWOCons

Input: The secret key $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$ and index (i,j,k).

Output: $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$.

```
1: $\mathcal{L}_{i,j,k} \leftarrow \mathsf{DGMT.Shuffle}(\mathsf{DGMT.SK}, (i,j,k))$;
2: for $0 \le l \le \alpha - 1$ do
3:     $\mathsf{DGMT.pos}_{i,j,k,l} \leftarrow \mathsf{g}_2(\mathsf{msk}, i \,\|\, j \,\|\, k \,\|\, l)$;
4:     $l' = \mathcal{L}_{i,j,k}[l]$;
5:     $\mathsf{OTS.sk}_{i,j,k,l'} \leftarrow \mathsf{f}(\mathsf{SMT_2.key}, i \,\|\, j \,\|\, k \,\|\, l')$;
6:     Compute $\mathsf{OTS.pk}_{i,j,k,l'}$ from $\mathsf{OTS.sk}_{i,j,k,l'}$;
7:     $h_{l'} \leftarrow \mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \,\|\, \mathsf{DGMT.pos}_{i,j,k,l})$;
8: end for
9: Construct a Merkle tree using $\{h_{l'}\}_{l'=0}^{\alpha-1}$, called $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$;
10: return $\mathsf{SMT}^{(2)}_{\mathsf{i,j,k}}$;
```

4.3 DGMT's Algorithms

In Subsection 3, we presented the algorithms of a $\mathcal{FDGS}$. Here, we describe all the algorithms of DGMT in detail, as a $\mathcal{FDGS}$.

1. $(\mathsf{DGMT.SK}, \mathsf{DGMT.PubPr}) \leftarrow \mathsf{DGMT.KG}(1^\lambda, \mathsf{DGMT.SetPr})$: The manager runs algorithm 4 on the security parameter λ and the setup parameters $\mathsf{DGMT.SetPr} = (h_I, h_{SM}, \gamma, \mathsf{T}_{\mathsf{tot}}, \mathsf{N}_{\max})$ to obtain $(\mathsf{DGMT.SK}, \mathsf{DGMT.PubPr})$, where $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$ and $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$. These values are generated as follows:
   - $\mathsf{msk} \xleftarrow{\$} \{0,1\}^\lambda$ is the manager's master secret key of the SPRP g_2 that is used for opening signatures, revoking users, and generating $\mathsf{SMT}^{(2)}$s.

- $\mathsf{IMT.key} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ is the secret key of the PRF f for generating an IMT of height h_I, where its root, i.e. r_{IMT}, is the group public key $\mathsf{DGMT.gpk}$.
- $\mathsf{SMT_1.key} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ is the secret key of the PRF f for generating the $\mathrm{SMT}^{(1)}$s on the OTS key pairs $(\mathsf{OTS.sk}_{i,j,k}, \mathsf{OTS.pk}_{i,j,k})$, with $1 \le i < 2^{h_I+1} - 1$, $1 \le j \le \gamma$ and $1 \le k \le \alpha$.
- $\mathsf{SMT_2.key} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ is the secret key of the PRF f for generating the $\mathrm{SMT}^{(2)}$s on the OTS key pairs $(\mathsf{OTS.sk}_{i,j,k,l}, \mathsf{OTS.pk}_{i,j,k,l})$s, with $1 \le i < 2^{h_I+1} - 1$, $1 \le j \le \gamma$ and $1 \le k, l \le \alpha$.
- $\mathsf{shuffle.key} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ is the secret key of the SPRP g_1 for shuffling the leaf nodes of $\mathrm{SMT}^{(2)}$s.
- $\mathcal{FK} = [Fk_{i,j} \leftarrow \mathsf{g}_1(r_{i,j}, Fn_i) \mid 1 \le i < 2^{h_I+1} - 1,\ 1 \le j \le \gamma]$.
- $\mathcal{RL}$ is the revocation list that is initially empty.

Algorithm 4 $\mathsf{DGMT.KG}$

```
Input: The security parameter $\lambda$ and $\mathsf{DGMT.SetPr} = (h_I, h_{SM}, \gamma, \mathsf{T_{tot}}, \mathsf{N_{max}})$.
Output: $(\mathsf{DGMT.SK}, \mathsf{DGMT.PubPr})$

1: $\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key} \overset{\$}{\leftarrow} \{0,1\}^\lambda$;
2: $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathsf{IMT.key}, \mathsf{SMT_1.key}, \mathsf{SMT_2.key}, \mathsf{shuffle.key})$;
3: $\mathsf{DGMT.PubPr} \leftarrow \mathsf{DGMT.PubPrCons}(1^\lambda, \mathsf{DGMT.SetPr}, \mathsf{DGMT.SK})$;
4: return $(\mathsf{DGMT.SK}, \mathsf{DGMT.PubPr})$;
```

2. $((\mathsf{PL}_\mathcal{M}, \mathsf{ID}), (\mathsf{id}, c_{\mathsf{id}}))/\perp \leftarrow \mathsf{DGMT.Join}(\mathsf{Username})$: This interactive joining protocol, the same as $\mathsf{FDGS.Join}$ in Section 3.1, occurs between the manager $\mathcal{M}$ and a prospective user with identity $\mathsf{Username}$ over a secure channel. If $\mathsf{Username} \notin \mathsf{ID}$ and $|\mathsf{ID}| < \mathsf{N_{max}}$, the manager $\mathcal{M}$ allocates the smallest unassigned identifier id with $1 \le \mathsf{id} \le \mathsf{N_{max}}$ along with a secret value c_{id} to this user and sends $(\mathsf{id}, c_{\mathsf{id}})$ to the user. Otherwise, the algorithm outputs $\perp$. The manager $\mathcal{M}$ stores $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$

Algorithm 5 $\mathsf{DGMT.Join}$

```
Input: The identity $\mathsf{Username}$ of a user
Output: $(\mathsf{id}, c_{\mathsf{id}})$

1: The user sends $\mathsf{Username}$ along with a request to join the group;
2: The manager $\mathcal{M}$ takes the next available unassigned $\mathsf{id} \le \mathsf{N_{max}}$;
3: $\mathcal{M}$ generates $c_{\mathsf{id}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$;
4: $\mathcal{M}$ sends $(\mathsf{id}, c_{\mathsf{id}})$ to $\mathsf{Username}$;
5: $\mathcal{M}$ appends $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$ to the list $\mathsf{PL}_\mathcal{M}$;
6: $\mathcal{M}$ appends $\mathsf{Username}$ to the id-th position in the list $\mathsf{ID}$.
7: $\mathsf{Username}$ stores the received $(\mathsf{id}, c_{\mathsf{id}})$;
```

in the private list $\mathsf{PL}_\mathcal{M}$ and the user's identity $\mathsf{Username}$ as the id-th element in the list ID, where ID is the list of the users' identities who have already joined the group and is initially empty. See Algorithm 5 for details.
When a user with identifier id requires new OTS key pairs, they initiate Subroutine 6 by sending a request to the manager via the Algorithm $\mathsf{DGMT.OTSReq}(\mathsf{id}, c_{\mathsf{id}})$. Upon receiving the request, the manager allocates B new private keys $\{\mathsf{gsk}^{\mathsf{id}}_{i_r,j,k,l'}\}_{r=1}^B$ to the user by executing the Algorithm $\mathsf{DGMT.KeyDist}(\mathsf{id})$, provided that $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active}) \in \mathsf{PL}_\mathcal{M}$, i.e. the user has already joined the group and has not been revoked.
To execute the key allocation effectively, the manager assigns each user with identifier $\mathsf{id} \in [1, \mathsf{N_{max}}]$ a list:

$$\mathsf{Index_{id}} = [(i, (j,k,l)) \mid 1 \le i < 2^{h_I} - 1].$$

This list corresponds to the internal nodes of the IMT, where the second component of the i-th entry, i.e. (j,k,l), represents the last signing key assigned to the user id from the $\mathrm{SMT^{MT}}$ linked to the internal node Fn_i. In other words, the entry $(i,(j,k,l)) \in \mathsf{Index_{id}}$ indicates that user id has received their l-th signing key from $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$. Initially, this list is set as $\mathsf{Index_{id}} = [(i,(1,0,0)) \mid 1 \le i < 2^{h_I} - 1]$, and is updated after each allocation. Algorithm $\mathsf{DGMT.KeyDist}$ explains the process of selecting the next B available indexes for the user id and updating $\mathsf{Index_{id}}$ accordingly.

Remark 4 In lines 5–9 of Algorithm $\mathsf{DGMT.KeyDist}$, we maintain the bound $l \le \beta - 2$ to ensure that every user has at least one unused key in each $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$, as required by the anonymity proof.

3. $\sigma_{\mathrm{DGMT}} \leftarrow \mathsf{DGMT.Sig}(m, \mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'})$: The signing algorithm takes as input a message m and an unused private key $\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}$ that belongs to the user id and outputs a signature σ_{DGMT} as explained in Algorithm 7.

Remark 5 In Algorithm 7, the depth μ of the internal node Fn_i in the IMT is used. The use of fixed-height $\mathrm{SMT^{MT}}$s results in variable-length signatures, and including μ as part of the signed message helps prevent potential exploitation of this variability in signature length.

4. $0/1 \leftarrow \mathsf{DGMT.Vf}(m, \sigma_{\mathrm{DGMT}}, \mathsf{DGMT.PubPr})$: To verify a pair $(m, \sigma_{\mathrm{DGMT}})$, the verifier runs the deterministic Algorithm 8, which outputs 1 if and only if σ_{DGMT} is a valid signature on m.

Remark 6 In DGMT, having (i,j) allows the verifier to uniquely determine the fallback key $Fk_{i,j}$ from the public list $\mathcal{FK}$. Therefore, unlike DGM where $Fk_{i,j}$ is a part of the signature, in DGMT $Fk_{i,j}$ is not a part of the signature.

Algorithm 6 Subroutines of Key Distribution of DGMT

```
DGMT.OTSReq:

Input: A pair $(\mathsf{id}, c_{\mathsf{id}})$ and $\mathsf{Index}_{\mathsf{id}}$
Output: $\{\mathsf{gsk}^{\mathsf{id}}_{i_r,j,k,l'}\}_{r=1}^{B}$ or $\perp$.

1: if $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active}) \notin \mathsf{PL}_{\mathcal{M}}$ then;
2:     return $\perp$ ;
3: end if
4: $\{\mathsf{gsk}^{\mathsf{id}}_{i_r,j,k,l'}\}_{r=1}^{B} \leftarrow \mathsf{DGMT.KeyDist}(\mathsf{id})$
5: return $\{\mathsf{gsk}^{\mathsf{id}}_{i_r,j,k,l'}\}_{r=1}^{B}$

DGMT.KeyDist:

Input: An identifier id.
Output: $B$ new keys $\{\mathsf{gsk}^{\mathsf{id}}_{i_r,j,k,l'}\}_{r=1}^{B}$.

1: $\mathsf{count} \leftarrow 1$;
2: while $(\mathsf{count} \leq B)$ do
3:     Randomly choose an internal node $Fn_i$;
4:     $(j,k,l) \leftarrow$ The second component of the $i$-th tuple from $\mathsf{Index}_{\mathsf{id}}$;
5:     if $l < \beta - 2$, then $(j,k,l) \leftarrow (j,k,l+1)$;
6:     else if $l = \beta - 2 \wedge k < \alpha - 1$, then $(j,k,l) \leftarrow (j,k+1,0)$;
7:     else if $l = \beta - 2 \wedge k = \alpha - 1 \wedge j < \gamma$, then $(j,k,l) \leftarrow (j+1,0,0)$;
8:     else goto Step 3;
9:     end if
10:    if $k = 0$, then construct the $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$ if does not exist;
11:    end if
12:    if $l = 0$, then construct the $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$ if does not exist;
13:    end if
14:    $l' \leftarrow \mathcal{L}_{i,j,k}[l + (\mathsf{id}-1)\beta]$;
15:    Retrieve the $l'$-th signing key from $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$,

$$\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'} = ((i,j,k,l'), \mathsf{OTS.sk}_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i); \quad (2)$$

       //where,
16:        //$(i,j,k,l')$: Index of the signing OTS;
17:        //$\mathsf{OTS.sk}_{i,j,k,l'}$: OTS secret key of the leaf node at index $(i,j,k,l')$;
18:        //$\mathsf{OTS.pk}_{i,j,k,l'}$: OTS public key of the leaf node at index $(i,j,k,l')$;
19:        //$\mathsf{DGMT.pos}_{i,j,k,l}$: $\mathsf{g}_2(\mathsf{msk}, i \parallel j \parallel k \parallel l)$;
20:        //$A.path_{i,j,k,l'}$: Authentication path from leaf node at index $(i,j,k,l')$ to the root $r_{i,j,k}$ of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$;
21:        //$\mathsf{OTS.sk}_{i,j,k}$: OTS secret key of the leaf node at index $(i,j,k)$;
22:        //$\mathsf{OTS.pk}_{i,j,k}$: OTS public key of the leaf node at index $(i,j,k)$;
23:        //$\sigma_{i,j,k}$: OTS signature on $r_{i,j,k}$ using $\mathsf{OTS.sk}_{i,j,k}$;
24:        //$A.path_{i,j,k}$: Authentication path from leaf node at index $(i,j,k)$ to the root $r_{i,j}$ of $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$;
25:        //$A.path_i$: Authentication path from the fallback node $Fn_i$ to the root of IMT;
26:    $\mathsf{count} \leftarrow \mathsf{count} + 1$;
27: end while
28: return $\{\mathsf{gsk}^{\mathsf{id}}_{i_r,j,k,l'}\}_{r=1}^{B}$
```

Algorithm 7 DGMT.Sig

```
Input: A message $m$ and an unused signing key $\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}$.
Output: $\sigma_{\mathrm{DGMT}}$

1: Parse $\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}$ as

$$\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'} = ((i,j,k,l'), \mathsf{OTS.sk}_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i);$$

2: Compute the height $\mu$ of $Fn_i$ in the IMT from $i$;
3: $m' \leftarrow \mathsf{H}(m \| \mu)$;
4: $\sigma_{i,j,k,l'} \leftarrow \mathsf{OTS.Sig}(\mathsf{OTS.sk}_{i,j,k,l'}, m')$;
5: Compute $\sigma_{\mathrm{DGMT}}$ as

$$\sigma_{\mathrm{DGMT}} = ((i,j,k,l'), \sigma_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i); \quad (3)$$

6: if $\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}$ was the last unused key of user id, then
7:     user id requests for new keys by the Algorithm $\mathsf{DGMT.OTSReq}(\mathsf{id}, c_{\mathsf{id}})$;
8: end if
9: return $\sigma_{\mathrm{DGMT}}$;
```

Algorithm 8 DGMT.Vf

```
Input: message $m$, signature $\sigma_{\mathrm{DGMT}}$, and $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$.
Output: 0/1

1: Parse $\sigma_{\mathrm{DGMT}}$ as

$$\sigma_{\mathrm{DGMT}} = ((i,j,k,l'), \sigma_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i);$$

2: if $\mathsf{DGMT.pos}_{i,j,k,l} \in \mathcal{RL}$, then return 0;
3: end if
4: Compute the height $\mu$ of $Fn_i$ in the IMT from $i$;
5: $m' \leftarrow \mathsf{H}(m \| \mu)$;
6: if $\mathsf{OTS.Vf}(m', \sigma_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}) = 0$, then return 0;
7: end if
8: $h' \leftarrow \mathsf{H}\left(\mathsf{OTS.pk}_{i,j,k,l'} \parallel \mathsf{DGMT.pos}_{i,j,k,l}\right)$;
9: Compute $r'_{i,j,k}$, as the root of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$, using the index $(i,j,k,l')$, $h'$, and $A.path_{i,j,k,l'}$;
10: if $\mathsf{OTS.Vf}(r'_{i,j,k}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}) = 0$, then return 0;
11: end if
12: Compute $r'_{i,j}$, as the root of $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$, using the index $(i,j,k)$, $\mathsf{OTS.pk}_{i,j,k}$, and $A.path_{i,j,k}$;
13: $Fn'_i \leftarrow \mathsf{g}_1^{-1}(r'_{i,j}, \mathcal{FK}[\gamma(i-1)+j])$;
14: Compute $r'_{\mathrm{IMT}}$ from $i, Fn'_i$, and $A.path_i$;
15: if $r'_{\mathrm{IMT}} = \mathsf{DGMT.gpk}$ then return 1;
16: else return 0;
17: end if
```

5. $(\mathsf{PL}_{\mathcal{M}}, \mathsf{DGMT.PubPr}) \leftarrow \mathsf{DGMT.Rev}(\mathsf{DGMT.SK}, \mathsf{PL}_{\mathcal{M}}, \mathsf{DGMT.PubPr}, \mathsf{R})$: To revoke a set R of users, the manager runs Algorithm 9 to update both $\mathsf{PL}_{\mathcal{M}}$ and DGMT.PubPr. In particular, this algorithm changes the status of each $\mathsf{id} \in \mathsf{R}$ in $\mathsf{PL}_{\mathcal{M}}$, preventing them from requesting any further keys. See

Algorithm 9. For two given indexes (j_1, k_1, l_1) and (j_2, k_2, l_2), we have $(j_1, k_1, l_1) \leq (j_2, k_2, l_2)$ if $j_1 < j_2$, or $(j_1 = j_2 \wedge k_1 < k_2)$, or $(j_1 = j_2 \wedge k_1 = k_2 \wedge l_1 \leq l_2)$.

Algorithm 9 DGMT.Rev

```
Input: A list R, $\mathsf{PL}_{\mathcal{M}}$, $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathcal{FK}, \mathcal{RL})$, and $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$
Output: $(\mathsf{PL}_{\mathcal{M}}, \mathsf{DGMT.PubPr})$

1: for id $\in$ R do
2:     for $1 \leq i < 2^{h_I} - 1$ do
3:         Replace $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Active})$ with $(\mathsf{id}, c_{\mathsf{id}}, \mathsf{Revoked})$ in $\mathsf{PL}_{\mathcal{M}}$.
4:         $(j_i, k_i, l_i) \leftarrow$ Retrieve the second component of the $i$-th entry in $\mathsf{assign}_{\mathsf{id}}$;
5:         $\mathcal{RL} = \mathcal{RL} \cup \{\mathsf{g}_2(\mathsf{msk}, i \parallel j \parallel k \parallel l + (\mathsf{id} - 1)\beta) |$ for all $(j,k,l) \leq (j_i, k_i, l_i)\}$;
6:     end for;
7: end for;
8: $\mathsf{DGMT.PubPr} = (\mathsf{DGMT.gpk}, \mathcal{FK}, \mathcal{RL})$;
9: return $(\mathsf{PL}_{\mathcal{M}}, \mathsf{DGMT.PubPr})$;
```

6. $\mathsf{id}/\perp \leftarrow \mathsf{DGMT.Op}(\sigma_{\mathrm{DGMT}}, \mathsf{DGMT.SK})$: The manager runs the deterministic algorithm 10 to open the signature σ_{DGMT}. This algorithm outputs either the identifier id of the user who generated σ, or $\perp$ if the signature cannot be attributed to any specific user. As we already mentioned, the manager can retrieve the identity Username of the user by looking up the id-th entry in the list ID. See Algorithm 10.

Algorithm 10 DGMT.Op

```
Input: A signature $\sigma_{\mathrm{DGMT}}$ and $\mathsf{DGMT.SK} = (\mathsf{msk}, \mathcal{FK}, \mathcal{RL})$.
Output: $\mathsf{id}/\perp$

1: Extract $\mathsf{DGMT.pos}_{i,j,k,l}$ from $\sigma_{\mathrm{DGMT}}$;
2: $i \parallel j \parallel k \parallel l \leftarrow \mathsf{g}_2^{-1}(\mathsf{msk}, \mathsf{DGMT.pos}_{i,j,k,l})$;
3: if $(1 \leq i < 2^{h_I + 1} - 1) \wedge (0 \leq j \leq \gamma) \wedge (0 \leq k, l \leq \alpha - 1)$, then return $\mathsf{id} = \lceil \frac{l+0.5}{\beta} \rceil$;
4: else return $\perp$;
5: end if
```

4.4 Instantiating DGMT and Parameter Estimation

To compute the signature and public key size of DGMT, we use WOTS as the OTS. As argued in Remark 1, using WOTS allows us to omit $\mathsf{OTS.pk}_{i,j,k,l'}$ and $\mathsf{OTS.pk}_{i,j,k}$ from the DGMT signature and it will be as:

$$\sigma_{\mathrm{DGMT}} = ((i,j,k,l'), \sigma_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, A.path_{i,j,k}, A.path_i).$$

Now we compute the signature and public key sizes of DGMT in terms of the setup parameters of the system, security parameter λ and Winternitz parameter w.

Signature size:

- (i,j,k,l') is the index of the leaf node and is λ bits.
- $\sigma_{i,j,k,l'}$ and $\sigma_{i,j,k}$ are WOTS signature, each of size $\xi\lambda$ bits, where ξ is the number of elements in WOTS signature.
- $\mathsf{DGMT.pos}_{i,j,k,l}$ is the output of the SPRP g_2 and is λ bits.
- $A.path_{i,j,k,l'}, A.path_{i,j,k}$, and $A.path_i$ are the authentication paths in the $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$, $\mathrm{SMT}^{(1)}_{\mathrm{i,j}}$, and IMT, which is totally $(h_I + h_{SM})\lambda$ bits.

Thus, the signature size in bits, is bounded by

$$(h_I + h_{SM} + 2 + 2\xi)\lambda. \tag{4}$$

Public key size: The public key of DGMT is the root of the IMT, and is λ bits.

Parameter selection in our implementation: We consider a GSS with $\mathsf{T}_{\mathsf{tot}} = 2^{64}$ OTSs and $\mathsf{N}_{\mathsf{max}} = 2^{12}$ users, and choose the setup parameters $\mathsf{DGMT.SetPr} = (h_I, h_{SM}, \gamma, \mathsf{T}_{\mathsf{tot}}, \mathsf{N}_{\mathsf{max}}) = (16, 32, 2^{16}, 2^{64}, 2^{12})$. The configuration allows each user to receive up to 2^{52} OTSs from the manager[3]. The number of fallback keys is 2^{32} which can be generated comfortably during initialization. We note that, $\mathsf{N}_{\mathsf{max}}$ is bounded to $2^{h_S - 1}$ so having two-layer $\mathrm{SMT}^{\mathrm{MT}}$s with $h_{SM} = 32$ allows the system to accommodate 2^{15} users.

Let the security parameter $\lambda \in \{256, 384\}$ and $\omega = 4$. Given these parameters, Table 6 provides the signature size and public key size of DGMT. Theorem (1) shows that security of DGMT relies on the security of some symmetric-key primitives, especially the collision resistance of the underlying hash family. Thus classic and quantum security when λ is the output size of the hash function, are respectively $\lambda/2$ bits and $\lambda/3$ qbits. Therefore, for $\lambda = 256, 384$ its classic security is respectively 128 and 192 bits, and its quantum security is respectively 85 and 128 qbits [8, 31].

Remark 7 Security DGMT relies on the collision resistance of the hash function that is used to construct the Merkle trees (IMT and SMT^{MT}. Using the approach in [17] one can modify the trees so that the security of the resulting signature relies on the second pre-image resistance. This will allow the output size of the hash function to be halved for the same security level (the complexity of the best collision and preimage attacks for a hash function with output size λ is $2^{\lambda/2}$ and 2^{λ}, respectively), resulting in a shorter signature. This however requires careful analysis of the design because of *multi-target attack* that is applicable to the new design.

Multi-target attack was introduced in [30] in which the adversary makes d hash queries and succeeds if a second preimage for any of the queried values is found. It was

[3]This configuration provides exactly $\gamma(2^{h_I} - 2)2^{h_{SM}} = 2^{64} - 2^{48}$ OTSs.

Security parameter	$\xi_1 = \left\lceil \frac{\lambda}{w} \right\rceil$	$\xi_2 = \left\lfloor \frac{\log(\xi_1(2^w-1))}{w} \right\rfloor + 1$	$\xi = \xi_1 + \xi_2$	$\lvert\sigma_{DGMT}\rvert = (h_I + h_{SM} + 2 + 2\xi)\lambda$	$\lvert\mathsf{DGMT.gpk}\rvert = \lambda$
$\lambda = 256$	64	3	67	$(16+32+2+2\times 67)256$ bits = 5.75 KB	256 bits
$\lambda = 384$	96	3	99	$(16+32+2+2\times 99)384$ bits = 11.62 KB	384 bits

Table 6 Signature and public key sizes of DGMT for the given parameters.

shown that for a hash function with λ-bit output allowing d queries reduces the attack complexity from $\mathcal{O}(2^{\lambda})$ to $\mathcal{O}(2^{\lambda}/d)$ and so for λ-bit security either the hash function output size must be increased, or different keyed hash functions for each call must be used [30]. Adapting DGMT design to reduce security to second pre-image resistance will be our future work.

5 Security Proofs

In this section, we prove that DGMT provides Correctness, Unforgeability, Anonymity, and Traceability.

Theorem 1 *Let* DGMT *be the scheme described in Section 4. If (i)* H *is a randomly selected hash function from the collision-resistant hash function family* $\mathcal{HF}$*, (ii)* $\mathcal{OTS}$ *is an EU-CMA One-time Signature scheme, (iii)* $\mathsf{f} : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ *is a PRF, and (iv)* $\mathsf{g}_1 : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ *and* $\mathsf{g}_2 : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ *are two SPRPs, such that for any randomly selected tuple* (k,m,c) *with* $c \leftarrow \mathsf{g}_1(k,m)$ *the probability of finding a key* $k^* \neq k$ *with* $c \leftarrow \mathsf{g}_1(k^*,m)$ *is negligible, then* DGMT *satisfies Correctness, Unforgeability, Anonymity, and Traceability.*

Proof We follow the adversarial game model of [7, 12] to establish that DGMT satisfies Correctness, Unforgeability, Anonymity, and Traceability. Below we provide sketches of the proofs for each case.

Correctness: To prove correctness of DGMT, we show that if an adversary $\mathcal{A}$ corrupt all but one user, the honest user can successfully enroll and create signatures that are accepted by the verification algorithm and will be traced back to the user.

In the correctness game $\mathbf{Exp}^{\text{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$, a PPT adversary $\mathcal{A}$ can use oracles AddHU, and Revoke to add honest users and revoke the users, and AddCU to corrupt all but one user. In presence of such adversary, the honest user can still establish a secure channel to the (honest) group manager and securely receive $(\mathsf{id}, c_{\mathsf{id}})$. Only user and the group manager know $(\mathsf{id}, c_{\mathsf{id}})$ and the user uses $(\mathsf{id}, c_{\mathsf{id}})$ to receive $\{\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}\}_{i=1}^{B}$ from the manager using a secure challenge and generates correct signatures.

These keys are specifically allocated to this honest user, ensuring that for all $1 \le i \le B$, we have $l \in \mathcal{I}_{\mathsf{id}} = [(\mathsf{id}-1)\beta, \mathsf{id}\beta - 1]$, where (i,j,k,l') represents the index of the leaf node (i,j,k,l) after permutation by the SPRP g_1 (See Algorithm 2). Let $\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'} \in \{\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}\}_{i=1}^{B}$ and $\sigma_{DGMT} \leftarrow \mathsf{DGMT.Sig}(m, \mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'})$ be as

$$\sigma_{\text{DGMT}} = ((i,j,k,l'), \sigma_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, \\ A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i).$$

By computing

$$i \parallel j \parallel k \parallel l = \mathsf{g}_2^{-1}(\mathsf{msk}, \mathsf{DGMT.pos}_{i,j,k,l}), \tag{5}$$

the manager finds $l \in \mathcal{I}_{\mathsf{id}} = [(\mathsf{id}-1)\beta, \mathsf{id}\beta - 1]$, i.e. $(\mathsf{id}-1)\beta \le l \le \mathsf{id}\beta - 1$, so

$$\frac{\mathsf{id}\beta + (0.5 - \beta)}{\beta} \le \frac{l + 0.5}{\beta} \le \frac{\mathsf{id}\beta - 0.5}{\beta},$$

ensuring that computing $\left\lceil \frac{l+0.5}{\beta} \right\rceil$ yields the identifier id of the signer ($\lceil\cdot\rceil$ is the ceiling function.).

Unforgeability: To prove the unforgeability property of DGMT, we show that if an adversary $\mathcal{A}$ can forge a signature, then it creates a contradiction to the assumptions of the theorem.

In the unforgeability security game $\mathbf{Exp}^{\text{Unforg}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$, a challenger first runs the $\mathsf{DGMT.KG}$ to create the IMT/SMT$^{\text{MT}}$ trees and the secret and public keys of the system. Then the challenger gives access to the oracles AddHU, AddCU, SignHU, and Revoke to a PPT adversary $\mathcal{A}$. Therefore, $\mathcal{A}$ is allowed to obtain the signatures of every honest user on arbitrary messages. Each message-signature pair is stored in the signing list SL, preventing $\mathcal{A}$ from using these signatures as forgeries for their corresponding messages. $\mathcal{A}$ can revoke users, so their signatures will not be verified. Additionally, $\mathcal{A}$ can corrupt users to obtain their $(\mathsf{id}, c_{\mathsf{id}})$ and sign messages on their behalf, however, the generated signatures cannot be used as forgeries as those users are not honest.

We show that if $\mathcal{A}$ succeeds in forging a signature, i.e. it outputs a valid signature

$$\sigma^*_{DGMT} = ((i,j,k,l'), \sigma^*_{i,j,k,l'}, \mathsf{OTS.pk}^*_{i,j,k,l'}, \mathsf{DGMT.pos}^*_{i,j,k,l}, \\ A.path^*_{i,j,k,l'}, \sigma^*_{i,j,k}, \mathsf{OTS.pk}^*_{i,j,k}, A.path^*_{i,j,k}, A.path^*_i), \tag{6}$$

on a message m at the end of the query phase such that $\mathsf{id}^* \in \mathcal{H}$, where $\mathsf{id}^* \leftarrow \mathsf{DGMT.Op}(\sigma^*_{DGMT}, \mathsf{DGMT.SK})$, and $(m, \sigma^*_{DGMT}) \notin \mathsf{SL}$, then we can construct another PPT adversary $\mathcal{B}$ that simulates the role of the challenger for $\mathcal{A}$ to find a collision for the hash function H, or forge a valid signature for $\mathcal{OTS}$, or can find a second-key for g_1 that maps $Fk_{i,j}$ to Fn_i for some i, j. Note that in each case, one of the assumptions of the theorem will cause a contradiction.

Let σ^*_{DGMT} be a forged signature on message m. Since (m, σ^*_{DGMT}) is a valid signature then the index (i, j, k, l') of the σ^*_{DGMT} must be a valid index, i.e. $1 \le i < 2^{h_I+1} - 1$, $1 \le j \le \gamma$, and $0 \le k, l' \le \alpha - 1$. Let the index (i, j, k, l') belong to a user with identifier id. As $\mathcal{B}$ simulates the role of the challenger, it knows

$$\begin{aligned}\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'} =& ((i, j, k, l'), \mathsf{OTS.sk}_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'},\\ & \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k},\\ & A.path_{i,j,k}, A.path_i), \end{aligned} \tag{7}$$

and computes $\sigma_{DGMT} \leftarrow \mathsf{DGMT.Sig}(m, \mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'})$, where

$$\begin{aligned}\sigma_{\mathrm{DGMT}} = ((i, j, k, l'), \sigma_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l},\\ A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i).\end{aligned} \tag{8}$$

1. Let $A.path_i \ne A.path^*_i$. However σ_{DGMT} and σ^*_{DGMT} are both valid signatures for the index (i, j, k, l'). This implies that both paths lead to the same group public-key $\mathsf{DGMT.gpk}$. Therefore, $\mathcal{B}$ found a collision for H that contradicts the fact that H is a collision-resistant hash function.
2. Assume that $A.path_i = A.path^*_i$. Suppose that $A.path_{i,j,k}$ leads to $\mathrm{SMT}^{\mathrm{MT}}$ root node $r_{i,j}$ and $A.path^*_{i,j,k}$ leads to $\mathrm{SMT}^{\mathrm{MT}}$ root node $r^*_{i,j}$. In both cases, the $(\gamma(i-1)+j)$-th element of $\mathcal{FK}$, which is $Fk_{i,j}$, will be used to verify the signatures. Then we have $Fn_i \leftarrow \mathsf{g}_1^{-1}(r_{i,j}, Fk_{i,j})$ and $Fn^*_i \leftarrow \mathsf{g}_1^{-1}(r^*_{i,j}, Fk_{i,j})$. Now we have three cases.
   (i) Let $Fn_i \ne Fn^*_i$. Then $\mathcal{B}$ found a collision for H because $A.path_i = A.path^*_i$ and both of them leads to $\mathsf{DGMT.gpk}$.
   (ii) If $Fn_i = Fn^*_i$ and $r_{i,j} \ne r^*_{i,j}$. In this case, simulator $\mathcal{B}$ found two keys $r_{i,j} \ne r^*_{i,j}$ such that $FK_{i,j} \leftarrow \mathsf{g}_1(r_{i,j}, Fn_i)$ and $FK_{i,j} \leftarrow \mathsf{g}_1(r^*_{i,j}, Fn_i)$ which is a contradiction with the assumptions.
   (iii) If $Fn_i = Fn^*_i$ and $r_{i,j} = r^*_{i,j}$ which is discussed next.
3. If $(Fn_i, r_{i,j}, A.path_i) = (Fn^*_i, r^*_{i,j}, A.path^*_i)$. We have two cases:
   (i) If $A.path_{i,j,k} \ne A.path^*_{i,j,k}$. Since $r_{i,j} = r^*_{i,j}$, $\mathcal{B}$ found a collision for H and it forms a contradiction to the assumptions.
   (ii) Next we discuss the case when $A.path_{i,j,k} = A.path^*_{i,j,k}$.
4. If $A.path_{i,j,k} = A.path^*_{i,j,k}$ and $\mathsf{OTS.pk}_{i,j,k} \ne \mathsf{OTS.pk}^*_{i,j,k}$. There are three possible cases.
   (i) If $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k}) = \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k})$, $\mathcal{B}$ found a collision for H and it forms a contradiction to the assumptions.
   (ii) Let $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k}) \ne \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k})$. Since $A.path_{i,j,k} = A.path^*_{i,j,k}$ and $r_{i,j} = r^*_{i,j}$, $\mathcal{B}$ found a collision for the hash function H.
   (iii) Let $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k}) \ne \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k})$ and both the authentication path $A.path_{i,j,k}$ and $A.path^*_{i,j,k}$ leads to two different $\mathrm{SMT}^{\mathrm{MT}}$ root nodes $r_{i,j}$ and $r^*_{i,j}$ respectively. If $Fn_i \ne Fn^*_i$, then we are in case 2(i). If $Fn_i = Fn^*_i$, we are in the situation of case 2(ii).
5. If $A.path_{i,j,k} = A.path^*_{i,j,k}$ and $\mathsf{OTS.pk}_{i,j,k} = \mathsf{OTS.pk}^*_{i,j,k}$. There are two cases:
   (i) If $\sigma_{i,j,k} \ne \sigma^*_{i,j,k}$. This implies that $\mathcal{B}$ found a forgery for the $\mathcal{OTS}$ that contradicts the fact the $\mathcal{OTS}$ is EU-CMA secure.
   (ii) If $\sigma_{i,j,k} = \sigma^*_{i,j,k}$, which is discussed in the next case.
6. If $(\sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, Fn_i, r_{i,j}, A.path_i) = (\sigma^*_{i,j,k}, \mathsf{OTS.pk}^*_{i,j,k}, A.path^*_{i,j,k}, Fn^*_i, r^*_{i,j}, A.path^*_i)$. Assume that $A.path_{i,j,k,l'}$ leads to a $\mathrm{SMT}^{(2)}$ root node $r_{i,j,k}$ and $A.path^*_{i,j,k,l'}$ leads to a $\mathrm{SMT}^{(2)}$ root node $r^*_{i,j,k}$. We can divide this case into three subcases.
   (i) If $r_{i,j,k} \ne r^*_{i,j,k}$, then $\mathcal{B}$ found a forgery $(r^*_{i,j,k}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k})$ of $\mathcal{OTS}$ that contradicts to that the $\mathcal{OTS}$ is EU-CMA secure.
   (ii) Let $r_{i,j,k} = r^*_{i,j,k}$ and $A.path_{i,j,k,l'} \ne A.path^*_{i,j,k,l'}$. Then two different authentication paths of the same length h_S lead to the same $\mathrm{SMT}^{(2)}$ root node $r_{i,j,k}$. Therefore, $\mathcal{B}$ found a collision for the hash function H and hence a contradiction.
   (iii) If $r_{i,j,k} = r^*_{i,j,k}$ and $A.path_{i,j,k,l'} = A.path^*_{i,j,k,l'}$, which is discussed in the next case.
7. If $(A.path_{i,j,k,l'}, r_{i,j,k}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, Fn_{i,j}, r_{i,j}, A.path_i) = (A.path^*_{i,j,k,l'}, r^*_{i,j,k}, \sigma^*_{i,j,k}, \mathsf{OTS.pk}^*_{i,j,k}, A.path^*_{i,j,k}, Fn^*_{i,j}, r^*_{i,j}, A.path^*_i)$. There are two cases.
   (i) Let $(\mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}) \ne (\mathsf{OTS.pk}^*_{i,j,k,l'}, \mathsf{DGMT.pos}^*_{i,j,k,l})$. If $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \| \mathsf{DGMT.pos}_{i,j,k,l}) = \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k,l'} \| \mathsf{DGMT.pos}^*_{i,j,k,l})$, $\mathcal{B}$ found a collision for H and hence a contradiction.
   (ii) On the other hand, let $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \| \mathsf{DGMT.pos}_{i,j,k,l}) \ne \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k,l'} \| \mathsf{DGMT.pos}^*_{i,j,k,l})$. Since $A.path_{i,j,k,l'} = A.path^*_{i,j,k,l'}$ and $r_{i,j,k} = r^*_{i,j,k}$, $\mathcal{B}$ found a collision for H and that leads to a contradiction.
8. Lastly, we assume that all the components of σ_{DGMT} and σ^*_{DGMT} are the same except the components $\sigma_{i,j,k,l'}$ and $\sigma^*_{i,j,k,l'}$, that is $\sigma_{i,j,k,l'} \ne \sigma^*_{i,j,k,l'}$. Then $\mathcal{B}$ found a forgery for the for $\mathcal{OTS}$ that contradicts the EU-CMA security of the $\mathcal{OTS}$.

Therefore, we can conclude that DGMT achieves unforgeability.

Traceability: In traceability game $\mathbf{Exp}^{\mathrm{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$, the challenger runs the $\mathsf{DGMT.KG}$ to create the $\mathrm{IMT/SMT}^{\mathrm{MT}}$ tree and generate the secret and public keys of the system. The PPT adversary $\mathcal{A}$ has access to oracles AddHU, Revoke, AddCU, and SignHU oracles and can add and revoke users, corrupts them and obtain their $(\mathsf{id}, c_{\mathsf{id}})$, and obtains all signatures of honest users. At the end of the query phase, the adversary $\mathcal{A}$ has to output a valid signature

$$\sigma^*_{DGMT} = ((i,j,k,l'), \sigma^*_{i,j,k,l'}, \mathsf{OTS.pk}^*_{i,j,k,l'}, \mathsf{DGMT.pos}^*_{i,j,k,l}, A.path^*_{i,j,k,l'}, \sigma^*_{i,j,k}, \mathsf{OTS.pk}^*_{i,j,k}, A.path^*_{i,j,k}, A.path^*_i),$$

on a message m with $\mathsf{DGMT.Op}(\sigma^*_{DGMT}, \mathsf{DGMT.SK}) = \perp$. We will use this adversary to construct another PPT adversary $\mathcal{B}$ that simulates the role of the challenger for $\mathcal{A}$ to find a collision for the hash function H, or forge a valid signature for $\mathcal{OTS}$, or can find a second-key for g_1 that maps $Fk_{i,j}$ to Fn_i for some i, j. Note that in each case, one of the assumptions of the theorem will cause a contradiction.

Let σ^*_{DGMT} be a valid signature on m, with $\mathsf{DGMT.Op}(\sigma^*_{DGMT}, \mathsf{DGMT.SK}) = \perp$. The (i,j,k,l') is a valid index, so it belongs to a user with identifier id and the manager knows its corresponding $\mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'}$ given in (7) and is able to compute $\sigma_{DGMT} \leftarrow \mathsf{DGMT.Sig}(m, \mathsf{gsk}^{\mathsf{id}}_{i,j,k,l'})$, where

$$\sigma_{\mathrm{DGMT}} = ((i,j,k,l'), \sigma_{i,j,k,l'}, \mathsf{OTS.pk}_{i,j,k,l'}, \mathsf{DGMT.pos}_{i,j,k,l}, A.path_{i,j,k,l'}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i).$$

The proof arguments are the same as the proof presented for *Unforgeability*. The only difference is that $\mathsf{DGMT.pos}^*_{i,j,k,l} \neq \mathsf{DGMT.pos}_{i,j,k,l}$ in the traceability proof because $\mathsf{DGMT.Op}(\sigma^*_{DGMT}, \mathsf{DGMT.SK}) = \perp$ and $\mathsf{DGMT.Op}(\sigma_{DGMT}, \mathsf{DGMT.SK}) = \mathsf{id}$. Therefore, we also have $\mathsf{OTS.pk}^*_{i,j,k,l'} \parallel \mathsf{DGMT.pos}^*_{i,j,k,l} \neq \mathsf{OTS.pk}_{i,j,k,l'} \parallel \mathsf{DGMT.pos}_{i,j,k,l}$.

1. If $\mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k,l'} \parallel \mathsf{DGMT.pos}^*_{i,j,k,l}) = \mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \parallel \mathsf{DGMT.pos}_{i,j,k,l})$, then we found a collision for H. If not, then we have the next cases.
2. Let $\mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k,l'} \parallel \mathsf{DGMT.pos}^*_{i,j,k,l}) \neq \mathsf{H}(\mathsf{OTS.pk}_{i,j,k,l'} \parallel \mathsf{DGMT.pos}_{i,j,k,l})$, and $A.path_{i,j,k}$ and $A.path^*_{i,j,k}$ leads to $\mathrm{SMT}^{(2)}$ root nodes $r_{i,j,k}$ and $r^*_{i,j,k}$ respectively. If $r_{i,j,k} = r^*_{i,j,k}$, then $\mathcal{B}$ found a collision for H. Otherwise, this leads to the next case.
3. Let $r_{i,j,k} \neq r^*_{i,j,k}$. If $\mathsf{OTS.pk}_{i,j,k} = \mathsf{OTS.pk}^*_{i,j,k}$, then we found a forgery for the $\mathcal{OTS}$ scheme. However, if $\mathsf{OTS.pk}_{i,j,k} \neq \mathsf{OTS.pk}^*_{i,j,k}$ then there are two cases:
(i) If $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k}) = \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k})$, thus $\mathcal{B}$ found a collision for H.
(ii) If $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k}) \neq \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k})$, we have the next case.
4. Let $\mathsf{H}(\mathsf{OTS.pk}_{i,j,k}) \neq \mathsf{H}(\mathsf{OTS.pk}^*_{i,j,k})$, and $A.path_{i,j}$ and $A.path^*_{i,j}$ leads to $\mathrm{SMT}^{\mathrm{MT}}$ root nodes $r_{i,j}$ and $r^*_{i,j}$ respectively. If $r_{i,j} = r^*_{i,j}$, then $\mathcal{B}$ found a collision for H. Next, we consider the case $r_{i,j} \neq r^*_{i,j}$.
5. Let $r_{i,j} \neq r^*_{i,j}$. Assume that $Fn_i = \mathsf{g}_1^{-1}(r_{i,j}, FK_{i,j})$ and $Fn^*_i = \mathsf{g}_1^{-1}(r^*_{i,j}, FK_{i,j})$. If $Fn_i = Fn^*_i$, then the adversary $\mathcal{B}$ can find a new key $r^*_{i,j}$ such that $\mathsf{g}_1(r_{i,j}, Fn_i) = \mathsf{g}_1(r^*_{i,j}, Fn_i)$. Otherwise, that is if $Fn_i \neq Fn^*_i$, then $\mathcal{B}$ can find a collision for H using the authentication paths $A.path_i$ and $A.path^*_i$ because both the paths lead to the group public key $\mathsf{DGMT.gpk}$.

Hence, we achieve traceability.

Anonymity: We show that if $\mathcal{A}$ succeeds with a non-negligible advantage in distinguishing a signature generated by randomly selecting one of the two honest users with identifiers id_0 and id_1, then we can distinguish the outputs of the SPRP g_1 or g_2 from random permutations with non-negligible probability, which is a contradiction with our assumptions

In the anonymity security game $\mathbf{Exp}^{\mathrm{Anon-b}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$, a PPT adversary $\mathcal{A}$ has access to the oracles AddHU, AddCU, Revoke, SignHU, Ch_b, and Open. The adversary $\mathcal{A}$ is allowed to add honest users to the group and to revoke users from the group. It also can corrupt honest users to obtain their $(\mathsf{id}, c_{\mathsf{id}})$s and sign messages on their behalf. Additionally, $\mathcal{A}$ can use the oracle SignHU to obtain signatures from honest users on arbitrary messages. Furthermore, $\mathcal{A}$ can use the oracle Open to find the identity of the signer for any message-signature pair.

At some stage of the game, $\mathcal{A}$ calls Ch_b. $\mathcal{A}$ selects a message m along with two users with identifiers id_0 and id_1 and send them to the challenger. The challenger checks if id_0 and id_1 are honest or not. If they are honest, the challenger selects randomly a $\mathrm{SMT}^{(2)}$, say $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$, and a random bit b. According to Algorithm $\mathsf{DGMT.KeyDist}$ in Algorithm 6, every user has at least one unused key in every $\mathrm{SMT}^{(2)}$. So, the manager can select two unused keys of the users with identifiers id_0 and id_1 in $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$. Let (i,j,k,l'_0) and (i,j,k,l'_1) be the indexes of the leaf nodes of these unused keys which belong to the users id_0 and id_1, respectively (i.e. these two unused keys are $\mathsf{gsk}^{\mathsf{id}_0}_{i,j,k,l'_0}$ and $\mathsf{gsk}^{\mathsf{id}_1}_{i,j,k,l'_1}$). The challenger generates (m, σ^b_{DGMT}), where $\sigma^b_{DGMT} \leftarrow \mathsf{DGMT.Sig}(m, \mathsf{gsk}^{\mathsf{id}_b}_{i,j,k,l'_b})$ is

$$\sigma^b_{DGMT} = ((i,j,k,l'_b), \sigma_{i,j,k,l'_b}, \mathsf{OTS.pk}_{i,j,k,l'_b}, \mathsf{DGMT.pos}_{i,j,k,l_b}, A.path_{i,j,k,l'_b}, \sigma_{i,j,k}, \mathsf{OTS.pk}_{i,j,k}, A.path_{i,j,k}, A.path_i),$$

with $\mathsf{DGMT.pos}_{i,j,k,l_b} \leftarrow \mathsf{g}_2(\mathsf{msk}, i \parallel j \parallel k \parallel l_b)$, and sends it back to the adversary $\mathcal{A}$ to determine whose signature it is. $\mathcal{A}$ has still access with oracles AddHU, Revoke, SignHU, Ch_b, and Open. However, $\mathcal{A}$ cannot call the oracle Open on the signatures generated by Ch_b, which are stored in the challenge list CL. Also, $\mathcal{A}$ cannot invoke the oracle Revoke on users with identifiers id_0 or id_1, as this would allow $\mathcal{A}$ to distinguish the signer by searching $\mathsf{DGMT.pos}_{i,j,k,l_b}$ in the revocation list before and after calling the oracle Revoke on one of these identifiers. Finally, $\mathcal{A}$ outputs a bit b' and wins the anonymity experiment if $b' = b$.

Notice that (i, j, k, l'_b) and $\mathsf{DGMT.pos}_{i,j,k,l_b}$ are the only elements of the signature σ^b_{DGMT} that allows $\mathcal{A}$ to distinguish the signer. We discuss both cases separately below.

1. *$\mathcal{A}$ breaks the anonymity with non-negligible probability using (i, j, k, l'_b):*
   With the provided oracles, the adversary $\mathcal{A}$ can request signatures on arbitrary messages using all the leaf nodes of the selected $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$, except for the leaf nodes indexed by (i, j, k, l'_0) and (i, j, k, l'_1). Let (i, j, k, l^*) and (i, j, k, l^+) represent the tuples assigned to the leaf nodes indexed by (i, j, k, l'_0) and (i, j, k, l'_1) before permutation. According to the security model, $\mathcal{A}$ knows the values of $\mathsf{g}_1(\mathsf{shuffle.key}, i \parallel j \parallel k \parallel l)$, for all $l \neq l^*, l^+$. Consequently, $\mathcal{A}$ knows the permutation of (at most) all the leaf nodes of $\mathrm{SMT}^{(2)}_{\mathrm{i,j,k}}$ except two leaf nodes indexed by (i, j, k, l^*) and (i, j, k, l^+), and does not have any information on the values of $\mathsf{g}_1(\mathsf{shuffle.key}, i \parallel j \parallel k \parallel l^*)$ and $\mathsf{g}_1(\mathsf{shuffle.key}, i \parallel j \parallel k \parallel l^+)$. If $\mathcal{A}$ breaks the anonymity experiment by (i, j, k, l'_b) with non-negligible probability, it means that $\mathcal{A}$ is able to extract information on the two values $\mathsf{g}_1(\mathsf{shuffle.key}, i \parallel j \parallel k \parallel l^*)$ and $\mathsf{g}_1(\mathsf{shuffle.key}, i \parallel j \parallel k \parallel l^+)$, and identify which one would be located at the position (i, j, k, l'_b) with non-negligible probability. This would mean that $\mathcal{A}$ can distinguish the outputs of the SPRP g_1 from a random function with non-negligible probability, which leads to a contradiction.
2. *$\mathcal{A}$ breaks the anonymity with non-negligible probability using $\mathsf{DGMT.pos}_{i,j,k,l_b}$:*
   Using the notation of the previous case, $\mathsf{DGMT.pos}_{i,j,k,l_b}$ is either $\mathsf{g}_2(\mathsf{msk}, (i \parallel j \parallel k \parallel l^*))$ or $\mathsf{g}_2(\mathsf{msk}, (i \parallel j \parallel k \parallel l^+))$. Hence, if the adversary $\mathcal{A}$ breaks the anonymity experiment using $\mathsf{DGMT.pos}_{i,j,k,l_b}$ with non-negligible probability, it means that $\mathcal{A}$ is able to win the indistinguishability game of g_2 for two pairs $(i \parallel j \parallel k \parallel l^*)$ and $(i \parallel j \parallel k \parallel l^+)$ with non-negligible probability, which is a contradiction with our assumption.

It is important to note that although (i, j, k, l'_b) and $\mathsf{DGMT.pos}_{i,j,k,l_b}$ are both related to the index (i, j, k, l_b). The former represents the location of (i, j, k, l_b) after the permutation of the leaf nodes, determined by the SPRP g_1 and the secret key $\mathsf{shuffle.key}$ through the shuffling Algorithm 2. The latter is computed using the SPRP g_2 and the key msk. As a result, these values together do not reveal any information about the index (i, j, k, l_b) and equivalently the identifier of the signer. $\square$

6 Revocation in DGM and DGMT

One of the main contributions of DGM is the introduction of a new approach to revocation that uses $\mathcal{SPE}$ as its main building block. In the following, we review this approach and show that there is a significant hidden cost in using the approach, which makes the traditional revocation list a much better choice in practice for revocation. Moreover, in Appendix A, we describe how $\mathcal{SPE}$ decryption keys can be used to construct a revocation list, providing a more efficient approach to revocation than the original DGM. See Tabel 8 for the comparative study.

6.1 Revocation in DGM

DGM uses SPE-based approach to revocation. A DGM signature includes a unique tag that is the encryption of the index of the leaf node (called $\mathsf{DGM.pos}$) whose associated OTS has been used to sign the message. The verifier checks if the OTS key is revoked or not by checking whether it is getting back the message of the signature by first encrypting it using the master secret key and then decrypting the encrypted message using the punctured decryption key while using $\mathsf{DGM.pos}$ as the tag. If the OTS key is revoked, the decryption key SK_d is punctured at the tag $\mathsf{DGM.pos}$.

In the following, we use the description and parameters of $\mathcal{SPE}$ that was introduced in Section 2.1 to explain the revocation algorithm of DGM. Assume that there are d' punctured leaves in IMT/SMT structure at a certain point of time in DGM. If the manager wants to revoke a user to whom d'' signature positions are assigned, then the manager invokes the $\mathsf{SPE.KeyGen}$ algorithm with $d = d' + d''$ and receives the key $\mathsf{DGM}.rvk_0 = msk$. Next, the manager punctures the $\mathsf{DGM}.rvk_0$ at all the previous d' positions and new d'' positions by calling the $\mathsf{SPE.Punc}$ algorithm d times, and gets the punctured decryption key $\mathsf{DGM}.rvk_1 = SK_d$. The manager replaces the old keys with the new key pair $(\mathsf{DGM}.rvk_0, \mathsf{DGM}.rvk_1)$ and publishes them. Here $\mathsf{DGM}.rvk_0$ is the new encryption key of the $\mathcal{SPE}$, and $\mathsf{DGM}.rvk_1$ is the new decryption key. The verifier checks whether a signature is revoked by checking,

$$m \stackrel{?}{=} \mathsf{SPE.Dec}(\mathsf{DGM}.rvk_1, \mathsf{SPE.Enc}(\mathsf{DGM}.rvk_0, m, t), t), \tag{9}$$

where $t = \mathsf{DGM.pos}$ is the encrypted index of the leaf node associated with the signature and m is the received message. The tag space is the set of all possible leaf index. If the $\mathsf{DGM}.rvk_1$ is punctured at the position t, then the key of $\mathsf{SPE.Enc}$ derived from $(\mathsf{DGM}.rvk_0, t)$ is different from the key of $\mathsf{SPE.Dec}$ derived from $(\mathsf{DGM}.rvk_1, t)$ and thus comparison (9) will fail. For checking if a signature is revoked, the verifier performs two computational steps: SPE.Enc and SPE.Dec.

1. Encryption step SPE.Enc. Verifier first computes a chain of d keys: $sk_1, sk_2, \ldots, sk_d$ from DGM.$rvk_0 = msk = (sk_0, d)$. Then verifier applies the pseudorandom function F' on each of the $(d+1)$ sk_i for $i = 0, 1, \ldots, d$ along the tag t as described in Equation 9. DGM uses the construction of Pun-PRF in [29] and realizes SPE by constructing F' using GGM construction [25].

GGM-based construction of F'. Let G be a length doubling pseudorandom generator defined as $G : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{2\lambda}$. Let us denote the least significant λ bits and most significant λ bits of $G(seed)$ by $G_0(seed)$ and $G_1(seed)$ respectively. Let tag t be a λ-bit long binary-string $\{b_{\lambda-1} \cdots b_1 b_0\}$. We compute $F'(sk_i, t)$ as

$$F'(sk_i, t) = G_{b_{\lambda-1}}\left(\cdots G_{b_1}\left(G_{b_0}(sk_i)\right)\cdots\right).$$

We can view all executions of G as a binary tree of height λ where $G_0(seed)$ and $G_1(seed)$ are the right and left child, respectively, of the parent node *seed* and the tree is called GGM tree. Now the t-th leaf node of the GGM tree with root sk_i is the $F'(sk_i, t)$. Therefore, each $F'(sk_i, t)$ computation needs λ evaluations of G and, as a consequence, the verifier has to evaluate G $\lambda(d+1)$ times to compute the encryption key of SE.Enc oexcluding the Xors and computation of hash functions.

2. Decryption step SPE.Dec. During the computation of the decryption key with d punctured tags, the verifier needs to evaluate F.Eval d times and F' once on the key DGM.$rvk_1 = SK_d$ and the tag t. Let $SK_d = \{(sk_d, d), psk_1, psk_2, \ldots, psk_d\}$ which is punctured at tags $\{t_1, t_2, \ldots, t_d\}$. Each psk_i is the list of λ sibling nodes of the path that leads to the t_i-th leaf node of the GGM tree with the root sk_{i-1} and the j-th sibling node is

$$G_{\bar{b}_{i,j}}\left(G_{b_{i,j-1}}\left(\cdots G_{i,b_0}(sk_{i-1})\cdots\right)\right)$$

where t_i be a λ-bit binary string $t_i = \{b_{i,\lambda-1} \cdots b_{i,1} b_{i,0}\}$ and $\bar{b}_{i,j}$ is bit-wise complement of $b_{i,j}$. Therefore, each puncture or revocation needs λ evaluations of G and each puncture needs λ^2-bit storage. Notice that, given psk_i, we can compute any leaf node of the GGM tree with root sk_{i-1} except the leaf node at position t_i, that is we can compute F.Eval$(psk_i, t) = F'(sk_{i-1}, t)$ as long as $t \neq t_i$. Evaluation of F.Eval(psk_i, t) needs the sibling node in psk_i which is the root of the GGM subtree containing the t-th leaf node. Then, F.Eval(psk_i, t) needs at most $\lambda - 1$ and minimum 0 evaluations of G, that is, on average $(\lambda-1)/2$ evaluations of G is required. Therefore, the verifier has to perform on average $d(\lambda-1)/2 + \lambda$ evaluations of G to compute the decryption key of SE.Dec of SPE.Dec.

Total computation cost of revocation includes computation of $\mathcal{SPE}$ encryption key, evaluation of SE.Enc, computation of $\mathcal{SPE}$ decryption key, evaluation of SE.Dec, and string comparison which in total will be, $\lambda(d+1) + d(\lambda-1)/2 + \lambda = (3\lambda-1)d/2 + 2\lambda$ evaluations of G, one evaluation of SE.Enc and SE.Dec, and a string comparison on average.

6.2 Revocation in DGMT

In DGMT, for each revoked signature the manager appends DGMT.pos of the revoked signature to the revocation list $\mathcal{RL}$, and so after revoking d signatures, the revocation list will require $d\lambda$ bits of storage. For verification of a signature when d signatures are revoked, the verifier needs to, at the most, traverse a list of d elements with a string comparison at each element.

In comparison, for each revocation in DGM, the manager appends λ strings of length λ bits to the list DGM.rvk_1, and so when d signatures are revoked, DGM requires $d\lambda^2$ bits of storage. For verification of a signature, the verifier requires $(3\lambda-1)d/2 + 2\lambda$ evaluations of G, one evaluation of SE.Enc and SE.Dec, and a string comparison.

Thus for both computation and storage, the DGMT revocation list approach outperforms the DGM SPE-based approach.

7 Implementation and Experiments

In this section, we present our implementation and experimental results. Our implementation is for all the algorithms in Section 4. The implementation provides all the functions required in a fully dynamic symmetric-key based group signature. The code is available at https://github.com/submissionOfCode/DGMT_ref.

In our experiment, we use a system with OS Ubuntu 18.04, and the code is compiled using GCC-8.4.0. The processor of the system is Intel®Core™i7-9700 8-Core CPU @ 3.00GHZ and it has 8GB RAM. We use XMSS reference code, given in https://github.com/XMSS/xmss-reference, as the base of our software DGMT to comply with the fact the XMSS is a standard [15]. We use the AES functions in OpenSSL as a SPRP. We only use a single core without hyper-threading and turbo-boost. All the experimental results are listed in Table 7 and each reported timing is the average of 5 runs. Our experimental results show that DGMT is practical for large values of $\mathsf{T}_{\mathrm{tot}}$.

In the following, we explain the parameters chosen for our experiments. Consider the height of the IMT trees and setup time. For $\mathsf{T}_{\mathrm{tot}} = 2^{21}$ to 2^{27} signatures, we use an IMT of height $h_I = 4$ and SMT of height $h_S = 8, 9$, and 10. Thus the number of fallback keys is $(2^{h_I+1} - 2)\gamma = 30\gamma$ for $\gamma = 1, 2, 4$. For $\approx 2^{23}$ signatures, DGM needs an IMT with $h_I = 20$ and 524286 fallback keys which is significantly larger than the corresponding numbers in DGMT. Furthermore, DGM needs 4 seconds for the setup phase for $\approx 2^{23}$

Table 7 Timings for different Group Signature Operations, and Sizes of Signatures, IMT and different files in all the directories for different group parameters for $B = 8$. We use SHA-256 as the hash function and the Winternitz parameter w is 4. The column with $\gamma = 1$ is used for comparison for DGM. (s, ms, B, KB, and MB stand for seconds, Milliseconds, Bytes, Kilobytes, and Megabytes respectively).

Parameters									
h_I	4								
h_S (each layer of SMT$^{\text{MT}}$)	8			9			10		
β (leaf nodes/SMT$^{\text{MT}}_{\text{i,j,k}}$/user)	4			4			8		
γ (number of SMT$^{\text{MT}}$s per IMT node)	**1**	2	4	**1**	2	4	**1**	2	4
$\mathsf{N}_{\mathsf{max}}$ (group size)	$\mathbf{2^6}$	2^6	2^6	$\mathbf{2^7}$	2^7	2^7	$\mathbf{2^7}$	2^7	2^7
$\mathsf{T}_{\mathsf{tot}}$ (approx.)	$\mathbf{2^{20.54}}$	$2^{21.54}$	$2^{22.54}$	$\mathbf{2^{22.54}}$	$2^{23.54}$	$2^{24.54}$	$2^{24.76}$	$2^{25.76}$	$2^{26.76}$
Timings									
Setup (s)	**7.784**	15.412	30.843	**15.425**	31.179	61.765	**31.006**	61.639	123.259
Initial Key Distribution (s)	**862.080**	861.170	861.709	**5283.238**	4805.827	4508.836	**20679.256**	17393.360	20903.315
Joining (s)	**2.062**	2.058	2.060	**4.117**	4.139	4.116	**8.219**	8.222	8.529
User Signing (ms)	**0.468**	0.495	0.428	**0.429**	0.445	0.455	**0.402**	0.429	0.399
Verification before any revocation (s)	**1.026**	1.010	1.104	**1.161**	1.099	1.095	**1.123**	1.121	1.183
Revocation (Min) (ms)	**0.073**	0.074	0.073	**0.074**	0.073	0.074	**0.131**	0.135	0.134
Verification after revocation (Min) (s)	**1.045**	1.031	1.123	**1.180**	1.118	1.114	**1.160**	1.161	1.223
Opening (ms)	**0.0003**	0.0003	0.0003	**0.0003**	0.0003	0.0003	**0.0003**	0.0003	0.0003
Sizes									
Signature (average) (KB)	**5.216**	5.216	5.216	**5.280**	5.280	5.280	**5.344**	5.344	5.344
IMT size (B) (primary Memory)	**960**	960	960	**960**	960	960	**960**	960	960
FallBackKeys (KB)	**0.960**	1.920	3.840	**0.960**	1.920	3.840	**0.960**	1.920	3.840
Each file in m_user (B)	**240**	240	240	**240**	240	240	**240**	240	240
Each file in dgmt/manager/smt/smtU (B)	**132**	132	132	**132**	132	132	**132**	132	132
Each file in dgmt/manager/smt/smtD (MB)	**0.637**	0.637	0.637	**1.291**	1.291	1.291	**2.614**	2.614	2.614

There is a minor difference of ≈ 0.3 KB between the computed signature size of DGMT in (4), and the value shown in the table. This difference is because DGMT code uses XMSS code that incorporates additional data (e.g. index and public seed) in the signature. These date are not needed in DGMT.

signatures (Fig. 3 of [11]) while DGMT needs only 10 more extra seconds including the generation of all the fallback keys, and completely removes the interaction between the manager and the verifier. We have $\mathsf{N}_{\mathsf{max}} = 2^{h_S}/\beta$, where β is the total number of keys per user per SMT. We used $\beta = 4, 8$ and so $\mathsf{N}_{\mathsf{max}} = 2^{h_S-2}$ and 2^{h_S-3}. The following discussion provides the rationale for the results of our experiments. Joining time in DGMT depends on h_S and B only and does not depend on β and γ. This is because during the join operation of a user, for any γ, the first unassigned node of the user (from the B randomly chosen nodes of $\text{SMT}^{(2)}_{\text{i,j,k}}$) will be sent to the user. If h_S increases by one then the joining time increases by a factor of 2. Because an increase in h_S by one causes an increase in tree size by a factor of 2, and consequently the computation times of $\text{SMT}^{(1)}_{\text{i,j}}$ and $\text{SMT}^{(2)}_{\text{i,j,k}}$ increase 2 times. Notice that, user signing is of constant time but the verification time is variable and is larger than the user signing time. In a traditional MSS, the signing time includes the time required for computing both a WOTS and an authentication path. In DGMT however, the authentication paths are computed by the manager and sent privately to the user. Thus the user signing time is the time to compute a WOTS signature, and this takes a constant time. During verification, however, the verifier must check the signature against the revocation list to ensure it is not a revoked signature, and then recomputes the root of the IMT and compares it with the published root (public key). The time to check the signature against the revocation list depends on the size of the revocation list and the position of the corresponding encrypted label in the list, and for an unrevoked signature requires traversing the full revocation list. The recomputation time of the IMT root depends on the level of the fallback node in IMT and the h_S. Table 7 shows the average verification time for an empty revocation list, and when the B signatures of a single revoked user is included in the list. The signature opening is constant time because it only needs to decrypt the encrypted label which is a part of the signature. The revocation time depends on β and γ. For $h_S = 8, \mathsf{N}_{\mathsf{max}} = 2^6$, and $h_S = 9, \mathsf{N}_{\mathsf{max}} = 2^7$, so we have $\beta = 4$. For $h_S = 10, \mathsf{N}_{\mathsf{max}} = 2^7$, so $\beta = 8$. In our experiments, we revoke a user after the initial joining. Therefore, in the first two cases, we encrypt $\beta \times B = 32$ labels and add them to the revocation list. In third case, we have to encrypt and add 64 labels to the revocation list. Because of this, the revocation times for $h_S = 8, \mathsf{N}_{\mathsf{max}} = 2^6$, and $h_S = 9, \mathsf{N}_{\mathsf{max}} = 2^7$ are the same in Table 7, and $h_S = 10, \mathsf{N}_{\mathsf{max}} = 2^7$ shows doubled revocation time.

8 Concluding Remarks

We proposed DGMT, a post-quantum symmetric-key based fully dynamic group signature scheme that improves upon DGM by addressing two major shortcomings: (i) the need for interaction with the manager during signature verification, and (ii) the manager's storage size that grows proportional to the total number of signatures. We defined, formalized and proved DGMT and its security, and implemented and evaluated the scheme. Our experiments were conducted for $\mathsf{T}_{\mathsf{tot}} \approx 2^{24}$, but our design can support a much larger number of signatures: the flexible structure allows the setup parameters to be chosen for a total number $\mathsf{T}_{\mathsf{tot}}$ of OTSs and $\mathsf{N}_{\mathsf{max}}$ users, where $\mathsf{N}_{\mathsf{max}}$ is upper bounded by half of the number of leaves of $\mathrm{SMT}^{(2)}$.

An important advantage of DGMT over other post-quantum fully dynamic group signature schemes is that the signature length of DGMT is the shortest among all other known schemes with the same security level. Compared to SiTH, the only other fully implemented fully dynamic symmetric-key based group signature scheme, DGMT's signature size is approximately 100 times shorter (see Table 2).

Interesting directions for future work are: (i) design of more efficient stateless fully dynamic symmetric-key based group signature scheme, and (ii) adapting DGMT for applications that use EPID applications where large group sizes (e.g., 2^{60} that is supported by SiTH) are supported but traceability (signature opening) is not required.

Appendix A: A More Efficient SPE-based approach

We show that $\mathcal{SPE}$ decryption key DGM.rvk_1 can be used to construct a revocation list, which is much more efficient than the approach taken in DGM. Let the index of the revoked node for psk_i be $\hat{t} = \{b_{\lambda-1} \cdots b_1 b_0\}$. Then the j-th sibling node is $G_{\bar{b}_j}\left(G_{b_{j-1}}\left(\cdots G_{b_0}(sk_{i-1})\cdots\right)\right)$ for levels $j = 0, \ldots, \lambda - 1$. Now only the values of the sibling nodes do not convey whether a sibling node is a left child or right child of the parent and without that information, we cannot say which sibling node we need to compute $\mathsf{F.Eval}(psk_i, t)$. Therefore, we must associate an index with each sibling node and the j-th sibling node of psk_i must be $\left(G_{\bar{b}_j}\left(G_{b_{j-1}}\left(\cdots G_{b_0}(sk_{i-1})\cdots\right)\right), \bar{b}_j\right)$. Thus from the sibling nodes of psk_i, we can retrieve the λ-bit value of the tag $\hat{t}$ which is nothing but the $\mathsf{DGM.pos}$ and is also explicitly mentioned in the signature. Therefore, we can construct a revocation list containing d tags $\hat{t}_i$ corresponding to psk_i for $i = 1, \ldots, d$. In DGM, one can easily replace the SPE with the revocation list we just constructed. Therefore, every single revocation requires appending the $\mathsf{DGM.pos}$ of the revoked signature to the list. However, in the worst case, checking membership needs a traversal of the full list with d elements and d string comparisons. Table 8 summarizes our finding of the drawbacks of the SPE-based revocation method in DGM compared to the case of using a revocation list.

Appendix B: Reducing Setup Time

To reduce DGMT setup time, including the required time for generating the list $\mathcal{FK}$, one may increase the number of layers of $\mathrm{SMT}^{\mathrm{MT}}$ at the cost of increasing the signature length. For example, we can consider three-layer $\mathrm{SMT}^{\mathrm{MT}}$s where the topmost layers are short Merkle trees of height h_0. The number of fallback keys in this case will be $2^{64}/2^{32+h_0}$ and the computation of the list of fallback keys will be faster as the short topmost Merkle trees can be generated more quickly. As illustrated by the following example, the resulting increase in signature size is not overly significant. Setting $\lambda = 256$ and $\lambda = 384$ results in $\xi = 67$ and $\xi = 99$. Therefore, employing an $\mathrm{IMT}/\mathrm{SMT}^{\mathrm{MT}}$ structure with $h_{\mathrm{IMT}} = 16$ and $\mathrm{SMT}^{\mathrm{MT}}$s consisting of three layers and $h_{\mathrm{SMT}^{\mathrm{MT}}} = 33$ (where the topmost layer has height $h_0 = 1$ and the two lower layers each are of height 16) yields a signature size of $(h_I + h_{SM} + 2 + 3\xi)\lambda$ bits which are respectively $(16 + 33 + 2 + 3 \times 67)256 = 7.87$KB and $(16 + 33 + 2 + 3 \times 99)384 = 16.31$KB. Although this approach increases the DGMT signature size by 2KB to 5KB compared to the configuration given in Subsection 4.4, it significantly reduces the required setup time.

Author contribution The initial design of DGMT was proposed by Mojtaba Fadavi, who, along with Sabyasachi Karati and Reihaneh Safavi-Naini, contributed to its development, analysis, and the writing of this paper. The implementation of DGMT and related experiments were conducted by Sabyasachi Karati. Aylar Erfanian was involved in the project during the initial phase and contributed to the editing of the first version of the paper.

Funding The project was funded in part by Natural Sciences and Engineering Research Council of Canada under Industrial Chair Program supported by Telus Communications.

Declaration

Conflict of Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical Approval This study does not engage human participants or animals; hence, ethical issues pertaining to human or animal subjects are not relevant for this paper.

Table 8 Comparison between SPE-based revocation and Revocation list. GM stands for the manager.

	SPE	Revocation list (RL)
	Computation Cost	
Per revocation (GM)	λ evaluation of G	Addition of 1 encrypted OTS position or $\mathsf{DGM.pos}$ to the RL
Checking revoked or not (Verifier) (average case)	$(3\lambda-1)d/2+2\lambda$ evaluations of G +1 evaluation of $\mathsf{SE.Enc}$ +1 evaluation of $\mathsf{SE.Dec}$ +1 string comparison	$d/2$ string comparison
	Memory (in bits)	
Per revocation (GM)	λ^2	λ
Checking revoked or not (Verifier)	$d\lambda^2$	$d\lambda$

References

1. Alagic G., Apon D., Cooper D., Dang Q., Dang T., Kelsey J., Lichtinger J., Miller C., Moody D., Peralta R., et al.: Status report on the third round of the NIST post-quantum cryptography standardization process. US Department of Commerce, NIST (2022)
2. Bellare M., Micciancio D., Warinschi B.: Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. in Advances in Cryptology - EUROCRYPT 2003, vol. 2656, pp. 614–629, Springer, 2003. https://doi.org/10.1007/3-540-39200-9_38
3. Bellare M., Shi H., Zhang C.: Foundations of Group Signatures: The Case of Dynamic Groups. in Topics in Cryptology - CT-RSA 2005, vol. 3376, pp. 136–153, Springer, 2005.
4. Beullens W., Dobson S., Katsumata S., Lai Y., Pintore F.: Group signatures and more from isogenies and lattices: generic, simple, and efficient. Des. Codes Cryptogr., vol. 91, no. 6, pp. 2141–2200, 2023.
5. Beullens W., Kleinjung T., Vercauteren F.: CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations. in Advances in Cryptology - ASIACRYPT 2019, vol. 11921, pp. 227–247, Springer, 2019.
6. Boneh D., Eskandarian S., Fisch B.: Post-quantum EPID Signatures from Symmetric Primitives. in Topics in Cryptology - CT-RSA 2019, vol. 11405, pp. 251–271, Springer, 2019.
7. Bootle J., Cerulli A., Chaidos P., Ghadafi E., Groth J.: Foundations of Fully Dynamic Group Signatures. in Applied Cryptography and Network Security - ACNS 2016, vol. 9696, pp. 117–136, Springer, 2016.
8. Brassard G., Høyer P., Tapp A.: Quantum Cryptanalysis of Hash and Claw-Free Functions. in LATIN '98: Theoretical Informatics, Third Latin American Symposium, Campinas, Brazil, 1998, vol. 1380, pp. 163–169, Springer.
9. Brickell E. F., Camenisch J., Chen L.: Direct Anonymous Attestation. in Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington, USA, pp. 132–145, 2004.
10. Buchmann J., Dahmen E., Hülsing A.: XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. in Post-Quantum Cryptography - PQCrypto 2011, vol. 7071, pp. 117–129, Springer, 2011.
11. Buser M., Liu J. K., Steinfeld R., Sakzad A., Sun S.: DGM: A Dynamic and Revocable Group Merkle Signature. in Computer Security - ESORICS 2019, vol. 11735, pp. 194–214, Springer, 2019.
12. Camenisch J., Groth J.: Group Signatures: Better Efficiency and New Theoretical Aspects. in Security in Communication Networks - SCN 2004, vol. 3352, pp. 120–133, Springer, 2004.
13. Chaum D., van Heyst E.: Group Signatures. in Advances in Cryptology - EUROCRYPT '91, vol. 547, pp. 257–265, Springer, 1991.
14. 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), 11:1–11:35 (2024).
15. Cooper, D.A., Apon, D.C., Dang, Q.H., Davidson, M.S., Dworkin, M.J., Miller, C.A.: Recommendation for stateful hash-based signature schemes. NIST Special Publication, 800(208), 800–208 (2020).
16. Courtois N. T., Finiasz M., Sendrier N.: How to Achieve a McEliece-Based Digital Signature Scheme, in Advances in Cryptology - ASIACRYPT 2001, vol. 2248, pp. 157–174, Springer, 2001.
17. Dahmen E., Okeya K., Takagi T., Vuillaume C.: Digital Signatures Out of Second-Preimage Resistant Hash

Functions. in Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, vol. 5299, pp. 109–123, Springer, 2008.

18. Ducas L., Lepoint T., Lyubashevsky V., Schwabe P., Seiler G., Stehlé D.: CRYSTALS - Dilithium: Digital Signatures from Module Lattices. IACR Cryptol. ePrint Arch., vol. 2017, pp. 633, 2017. http://eprint.iacr.org/2017/633
19. El Bansarkhani R., Misoczki R.: G-Merkle: A Hash-Based Group Signature Scheme from Standard Assumptions. in Post-Quantum Cryptography - PQCrypto 2018, vol. 10786, pp. 441–463, Springer, 2018.
20. El Kaafarani A., Katsumata S., Solomon R.: Anonymous Reputation Systems Achieving Full Dynamicity from Lattices. in Financial Cryptography and Data Security - FC 2018, vol. 10957, pp. 388–406, Springer, 2018.
21. Esgin M. F., Steinfeld R., Zhao R. K.: MatRiCT^{+}: More Efficient Post-Quantum Private Blockchain Payments. in Proc. 43rd IEEE Symposium on Security and Privacy, San Francisco, USA, pp. 1281–1298, 2022.
22. ETSI: Quantum Safe Signatures, Standard ETSI TR 103 616 v1.1.
23. Ezerman M. F., Lee H. T., Ling S., Nguyen K., Wang H.: Provably Secure Group Signature Schemes From Code-Based Assumptions. IEEE Trans. Inf. Theory, vol. 66, no. 9, pp. 5754–5773, 2020.
24. Goldreich O.: Foundations of Cryptography. Volumes 2. Cambridge University Press, 2004.
25. Goldreich O., Goldwasser S., Micali S.: How to Construct Random Functions (Extended Abstract). in Proc. 25th Annual Symp. on Foundations of Computer Science, West Palm Beach, USA, pp. 464–479, 1984.
26. Gordon S. D., Katz J., Vaikuntanathan V.: A Group Signature Scheme from Lattice Assumptions. in Advances in Cryptology - ASIACRYPT 2010, vol. 6477, pp. 395–412, Springer, 2010.
27. Grover L. K.: A Fast Quantum Mechanical Algorithm for Database Search. in Proc. 28th Annual ACM Symp. on Theory of Computing, Philadelphia, USA, pp. 212–219, 1996.
28. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A Ring-Based Public Key Cryptosystem. In: Buhler, J. (ed.) Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer, (1998).
29. Hohenberger S., Koppula V., Waters B.: Adaptively Secure Puncturable Pseudorandom Functions in the Standard Model. in Advances in Cryptology - ASIACRYPT 2015, vol. 9452, pp. 79–102, Springer, 2015.
30. Hülsing, A., Rijneveld, J., Song, F.: Mitigating Multi-target Attacks in Hash-Based Signatures. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9614, pp. 387–416. Springer (2016).
31. Katz J., Kolesnikov V., Wang X.: Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. in Proc. 2018 ACM SIGSAC Conf. on Computer and Communications Security, Toronto, Canada, pp. 525–537, 2018.
32. Katz J., Lindell Y.: Introduction to Modern Cryptography. 2nd ed., CRC Press, 2014.
33. Hülsing A.: WOTS^{+} - Shorter Signatures for Hash-Based Signature Schemes. IACR Cryptol. ePrint Arch., vol. 2017, pp. 965, 2017. http://eprint.iacr.org/2017/965
34. Hülsing A., Butin D., Gazdag S., Rijneveld J., Mohaisen A.: XMSS: eXtended Merkle Signature Scheme. RFC, vol. 8391, pp. 1–74, 2018.
35. Hülsing A., Rausch L., Buchmann J.: Optimal Parameters for XMSS$^{\mathrm{MT}}$. in Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, vol. 8128, pp. 194–208, Springer, 2013.
36. Kiayias A., Yung M.: Extracting Group Signatures from Traitor Tracing Schemes. in Advances in Cryptology - EUROCRYPT 2003, vol. 2656, pp. 630–648, Springer, 2003.
37. Laguillaumie F., Langlois A., Libert B., Stehlé D.: Lattice-Based Group Signatures with Logarithmic Signature Size. in Advances in Cryptology - ASIACRYPT 2013, vol. 8270, pp. 41–61, Springer, 2013.
38. Lamport L.: Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, 1979.
39. Langlois A., Ling S., Nguyen K., Wang H.: Lattice-Based Group Signature Scheme with Verifier-Local Revocation. in Public-Key Cryptography - PKC 2014, vol. 8383, pp. 345–361, Springer, 2014.
40. Libert B., Ling S., Mouhartem F., Nguyen K., Wang H.: Signature Schemes with Efficient Protocols and Dynamic Group Signatures from Lattice Assumptions. in Advances in Cryptology - ASIACRYPT 2016, vol. 10032, pp. 373–403, Springer, 2016.
41. Ling S., Nguyen K., Wang H., Xu Y.: Lattice-Based Group Signatures: Achieving Full Dynamicity with Ease. in Applied Cryptography and Network Security - ACNS 2017, vol. 10355, pp. 293–312, Springer, 2017.
42. Ling S., Nguyen K., Wang H., Xu Y.: Constant-Size Group Signatures from Lattices. in Public-Key Cryptography - PKC 2018, vol. 10770, pp. 58–88, Springer, 2018.
43. Lyubashevsky V., Nguyen N. K., Plançon M., Seiler G.: Shorter Lattice-Based Group Signatures via 'Almost

Free' Encryption and Other Optimizations. in Advances in Cryptology - ASIACRYPT 2021, vol. 13093, pp. 218–248, Springer, 2021.

44. McGrew D. A., Curcio M., Fluhrer S. R.: Leighton-Micali Hash-Based Signatures. RFC, vol. 8554, pp. 1–61, 2019.
45. Merkle R. C.: A Certified Digital Signature. in Advances in Cryptology - CRYPTO '89, vol. 435, pp. 218–238, Springer, 1989.
46. Nguyen K., Tang H., Wang H., Zeng N.: New Code-Based Privacy-Preserving Cryptographic Constructions. in Advances in Cryptology - ASIACRYPT 2019, vol. 11922, pp. 25–55, Springer, 2019.
47. Rogaway P.: Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys. IACR Cryptol. ePrint Arch., vol. 2006, pp. 281, 2006. http://eprint.iacr.org/2006/281
48. Pass R., Shelat A.: A Course in Cryptography. 2017.
49. Shor P. W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. in Proc. 35th Annual Symp. on Foundations of Computer Science, Santa Fe, USA, pp. 124–134, 1994.
50. Sun S., Yuan X., Liu J. K., Steinfeld R., Sakzad A., Vo V., Nepal S.: Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption. in Proc. 2018 ACM SIGSAC Conf. on Computer and Communications Security, Toronto, Canada, pp. 763–780, 2018.
51. Yehia M., AlTawy R., Gulliver T. A.: GM^{MT}: A Revocable Group Merkle Multi-Tree Signature Scheme. in Cryptology and Network Security - CANS 2021, vol. 13099, pp. 136–157, Springer, 2021.
52. Yehia M., AlTawy R., Gulliver T. A.: Security Analysis of DGM and GM Group Signature Schemes Instantiated with XMSS-T. in Information Security and Cryptology - Inscrypt 2021, vol. 13007, pp. 61–81, Springer, 2021.