Dynamic Group Time-based One-time Passwords *

Xuelian Cao[1], Zheng Yang[1], Jianting Ning[2], Chenglu Jin[3], Rongxing Lu[4], Zhiming Liu[1], and Jianying Zhou[5]

[1] Southwest University, Chongqing, China
xueliancao7@email.swu.edu.cn, {youngzheng,Zhimingliu88}@swu.edu.cn
[2] Fujian Normal University, Fuzhou, China
jtning88@gmail.com
[3] Centrum Wiskunde & Informatica, Amsterdam, Netherlands
chenglu.jin@cwi.nl
[4] University of New Brunswick, Fredericton, Canada
rlu1@unb.ca
[5] Singapore University of Technology and Design, Singapore
jianying_zhou@sutd.edu.sg

Abstract. Group time-based one-time passwords (GTOTP) is a novel lightweight cryptographic primitive for achieving anonymous client authentication, which enables the efficient generation of time-based one-time passwords on behalf of a group without revealing any information about the actual client's identity beyond their group membership. The security properties of GTOTP regarding anonymity and traceability have been formulated in a static group management setting (where all group members should be determined during the group initialization phase), yet, a formal treatment for real-world dynamic groups (i.e., group members may join and leave at any time) is still an open question. It is non-trivial to construct an efficient GTOTP scheme that can provide a lightweight password generation procedure run by group members and support dynamic group management, allowing group members to join and leave without affecting other members' states (non-disruptively). To address the above challenge, we first define the notion and the security model of dynamic group time-based one-time passwords (DGTOTP) in this work. We then present an efficient DGTOTP construction that can generically transform an asymmetric time-based one-time passwords scheme into a DGTOTP scheme utilizing a chameleon hash function family and a Merkle tree scheme. Within our construction, we particularly tailor an outsourcing solution realizing an issue-first-and-join-later (IFJL) strategy, enabling smooth joining and revocation without disrupting other group members. Moreover, our scheme minimizes symmetric cryptographic operations and maintains constant storage for group members, compared to the linear storage cost that grows rapidly with respect to the lifetime of the GTOTP instance in the previous static GTOTP scheme. Our DGTOTP scheme satisfies stronger security guarantees in a dynamic group management setting without random oracles. Our experimental results confirm the efficiency of our DGTOTP scheme.

Keywords: Group Time-based One-Time Passwords · Dynamic Group Management · Anonymity · Traceability · Authentication · Security Model.

1 Introduction

Traditional user authentication relies on passwords, which are prone to hacking [26,34] and are not suitable for authenticating a client's device. To enhance the security of password-based authentication, Lamport [41] (hereinafter referred to as the Lamport'81 scheme) first proposed the cryptographic notion of one-time passwords (OTP), forcing users to frequently update their passwords for better security. To resist password reusing, traditional OTP methods incorporate time constraints to yield an important variant named time-based one-time passwords (TOTP), which is widely used in modern-day multi-factor authentication [27,22]. In a TOTP scheme, each password can only be used within a pre-determined time slot (e.g., 30 seconds). A standard TOTP (RFC 6238) [42] is implemented using traditional client and server shared secret keys and HMAC. This implies that, in the RFC 6238 standard scheme, the server must store the user password generation key in plaintext. If the server is compromised, the password seeds of all users will be leaked to the attacker, enabling further impersonations. Hence, a TOTP scheme that can resist server compromise is desirable in practice.

Applying asymmetric key distribution is a standard way to achieve server-compromise resilience in TOTP. That is, the server only stores the public verification key of the passwords instead of the password seeds. One classic asymmetric time-based one-time passwords construction method is the one-way function (OWF) chain structure introduced in the Lamport'81 scheme [41]. Each password is the seed (OWF pre-image) of the next password, i.e., $pw_i := \mathrm{OWF}(pw_{i-1})$. The last generated password is taken as the verification key, and the first password is used as the password seed for all other passwords. Most of the state-of-the-art asymmetric time-based one-time passwords algorithms adopt the framework of the Lamport'81 scheme, such as the T/Key [39]

* This is an extended version of the paper published at IEEE TIFS [11].

and the TOTP schemes [35] used in the constructions of proof of aliveness protocols. Since most of the existing TOTP schemes can be implemented with hash functions efficiently, they can survive the quantum attacks with an appropriate parameter setting [9,31,19]. This fact makes asymmetric TOTP schemes more appealing.

However, in traditional asymmetric TOTP schemes [39,35], each user in TOTP schemes has an independent password verification key. As a result, TOTP can provide no privacy of user identity. In practice, entities (e.g., mobile users and vehicles) in many applications (e.g., federated learning, indoor positioning, and V2X communication) usually need to protect their identity privacy when sharing information or requesting services. To protect user identity privacy in TOTP schemes, Yang *et al.* [56] recently proposed a cryptographic primitive named *group time-based one-time passwords* (GTOTP), allowing users to prove their group membership to arbitrary verifiers without revealing their real identity. It has been shown in [56] that GTOTP can be used as a useful tool to build an efficient privacy-preserving proof of location scheme (in which the distributed location witnesses and prover can jointly use GTOTP passwords for anonymous authentication). The seminal GTOTP scheme proposed by Yang *et al.* [56] (hereinafter referred to as YJN+ scheme) can generically transform an asymmetric TOTP scheme into a GTOTP scheme based on a permutation scheme and a Merkle tree structure, whose root is taken as the group public key. The Merkle tree leaves are bound to verification points of TOTP instances of group members, where each verification point is just used in a short time period called *verification epoch*. All leaves are randomly shuffled to achieve anonymity. However, the YJN+ scheme had certain limitations. It only considered a static group, namely, the group structure needed to be established during initialization. This assumption may not align with many real-world scenarios, such as business memberships and human resources management, where group members can join and leave at any time. Another shortcoming of the YJN+ scheme is its substantial storage overhead for group members. Suppose the YJN+ scheme is initialized with E verification epochs for U pre-determined group members. Then, it must involve $U \cdot E$ TOTP instances, which will be divided into ϕ groups for building ϕ Merkle trees. The roots of the generated Merkle trees are inserted into a Bloom filter, which is the final group public key. Therefore, the size of the group public key has a magnitude of $O(\phi)$, which is not a constant. The password of a group member consists of the password of the original TOTP instance and the corresponding Merkle proof, which is pre-stored by the group member. Namely, when we set $\phi = 1$ to have a constant size of the group public key, then each group member needs to store the Merkle proofs of all its verification points that have a size of $O(E \cdot \log^{U \cdot E})$. Thus, the storage cost grows rapidly when a long lifetime (large E) of the TOTP is expected.

Our Works. In this work, we take a significant step towards addressing the open question of building a GTOTP scheme in a dynamic group setting. However, designing an efficient dynamic time-based one-time passwords (DGTOTP) scheme with provable security guarantees is not a trivial task. In general, we have two main challenges. First, because group members may often use resource-constrained devices in practice, the algorithms used by group members in DGTOTP need to be efficient and avoid expensive public cryptographic operations. Otherwise, if public cryptographic operations can be run efficiently by group members, we can use more powerful cryptographic primitives (e.g., dynamic group signatures [8]) instead to realize the similar functionalities of DGTOTP. Second, the dynamic group management procedure for handling the joining and leaving operations of a group member should not affect the local states of all the other group members. This requirement is essential in many real-time applications (e.g., video surveillance and manufacturing execution) where group members cannot be interrupted by any group update procedures. Such a requirement poses a significant challenge in constructing a DGTOTP scheme.

To tackle the above challenges and requirements, we innovatively design an efficient DGTOTP scheme called DGTOne that provides provable security in our defined security model without random oracles. One of the novelties is that DGTOne only needs each group member to run a few symmetric cryptographic operations (e.g., pseudo-random function and hash function) and maintain a small constant-sized local state (either secret or public). We present a novel issue-first-and-join-later (IFJL) strategy based on the chameleon hash function to handle the join procedure and a creative outsourcing solution to realize efficient password generation. According to our IFJL strategy, a registration authority (RA), which takes charge of group management, first creates a Merkle tree using dummy verification points as leaves, and then issues independent secret seeds for joined group members on the fly. Each group member's secret seed is used as the chameleon hash function secret key (and many other secret keys) to dynamically bind a real verification point of a TOTP instance to a Merkle tree leaf during password usage. The generation of Merkle proofs and chameleon hash public keys are outsourced to RA, so each group member does not need to run any public key cryptographic operations. Moreover, such an outsourcing procedure results in a cost-free revocation feature. That is, to revoke a group member, RA only needs to stop generating chameleon hash public keys for it.

Furthermore, we also solve secure integration issues of DGTOTP passwords in practice. Unlike TOTP schemes, (D)GTOTP passwords are supposed to be used at different verifiers, i.e., the passwords should be publicly verifiable. However, a (D)GTOTP password itself does not provide any message authentication feature, so it may be subject to password replay attacks in an arbitrary-verifier setting, where some verifiers may reuse the received password to impersonate the password owner. Yang *et al.* [56] made the first attempt to solve this issue

in a commitment-based message authentication scenario that uses a GTOTP password to commit a message and open the password after it expires. One shortcoming of such a message authentication method is that its verification suffers a long delay in waiting for the commitment to open. To overcome this limitation, we aim to enhance the security integration of (D)GTOTP to realize a novel lightweight anonymous client authentication approach with low latency in an arbitrary-verifier setting with usage privacy.

Our Contributions. The contributions of this paper are summarized as follows:

- We define the notion and the security model of DGTOTP based on the framework of [56]. Compared with the previous work, our security model particularly formulates the threats regarding dynamic group management. In particular, we define new queries to simulate the joining of honest and malicious group members, and the revocation feature.
- We design an efficient DGTOTP scheme called DGTOne, leveraging a novel IFJL strategy we proposed. DGTOne provides provable security in our newly defined security model without random oracles.[6] In particular, our scheme requires no public key cryptographic operations for each group member. The storage cost maintained by each group member is a relatively small constant, instead of $O(E \cdot \log^{U \cdot E})$ in [56]. The revocation cost of DGTOne is optimal.
- We propose a security enhancement for resisting the password replay attacks in the traditional application scenario regarding passwords over TLS [18,43]. In our solution, we develop a password usage token (i.e., the hashes of a password and the verifier's identity, respectively) and use a semi-honest authentication server (AS) to record those password usage tokens for faithful-public retrieval. Meanwhile, the third-party AS only needs to store each token for a short time, so it will not incur much storage overhead. Thus, it can ensure that passwords are used at most by one verifier. In this way, a group member can instantly open the password in a server-only authenticated TLS channel (where the server can be the verifier) to complete the client's anonymous authentication.
- We evaluate the performance of DGTOne on a mobile phone. The results show that the password generation time is 119.5 microseconds on average. We also compare the performance of DGTOne with the YJN+ scheme based on the main operations. Although DGTOne provides more features for dynamic group management, the additional cost introduced to each group member is small and only consists of a few more modular additions and multiplications.

Organizations. Section 2 describes the preliminaries. Section 3 defines the security notions of DGTOTP. Section 4 introduces an efficient DGTOTP scheme. Section 5 discusses possible use cases. Section 6 shows the performance evaluation of our proposal. Section 7 presents the related work. We discuss privacy issues and a further comparison with anonymous client authentication schemes in Section 8. Section 9 concludes the paper.

2 Preliminaries

We denote the security parameter by κ, an empty string by $\emptyset$, and the set of integers between 1 and n by $[n] = \{1, \ldots, n\} \subset \mathbb{N}$. We let $x \xleftarrow{\$} X$ denote the operation of sampling x uniformly at random from a set X. Let $\mathsf{negl}(\kappa) : \mathbb{N} \to \mathbb{R}+$ be a negligible function, i.e., for every polynomial $P(\kappa)$ there exists a $e_0 \in \mathbb{N}$ s.t. for all $e > e_0$, $\mathsf{negl}(e) \leq 1/P(e)$. Let $\|$ be the string concatenation operation, and $\#$ be an operation to calculate the size of an element. In the following, we review the syntax and security definitions of the main cryptographic building blocks of our upcoming constructions.

Time-based One-time Passwords. We consider an asymmetric TOTP scheme which consists of four algorithms. $\mathsf{Setup}(1^\kappa, T_s, T_e, \Delta_s)$ takes as input the security parameter 1^κ, the start and end times T_s and T_e, and the password generation interval Δ_s, and outputs the password number $\mathsf{pms}_{\mathsf{TOTP}} = N = (T_e - T_s)/\Delta_s$. $\mathsf{PInit}(sd)$ takes as input a secret seed $sd \in \mathcal{K}_{\mathsf{TOTP}}$, and outputs the initial verification point $vp \in \mathcal{VP}_{\mathsf{TOTP}}$, where $\mathcal{K}_{\mathsf{TOTP}}$ is the key space for the input secret seed and $\mathcal{VP}_{\mathsf{TOTP}}$ is a verification point space. $\mathsf{PGen}(sd, T)$ takes as input the secret seed sd and a time slot T, and outputs a one-time password $pw \in \mathcal{PW}_{\mathsf{TOTP}}$ for T, where $\mathcal{PW}_{\mathsf{TOTP}}$ is a password space. $\mathsf{Verify}(vp, pw, T)$ takes as input the verification point vp, a password pw, and time slot T, and outputs 1 if the password is accepted and 0 otherwise. For tailoring the verification algorithm in our DGTOTP constructions, we require that the verification point of the TOTP scheme can be computed based on the corresponding secret seed sd or password. This requirement can be easily realized by any chain-based TOTP, such as [39,35]. We define a security game $G^{\mathsf{TOTP_Forge}}_{\mathcal{A},\mathsf{TOTP}}(\kappa, T_s, T_e, \Delta_s)$ for a time-based one-time passwords scheme TOTP in Figure 1, with a key space $\mathcal{K}_{\mathsf{TOTP}}$ for the input secret seed, a verification point space $\mathcal{VP}_{\mathsf{TOTP}}$, and a password space $\mathcal{PW}_{\mathsf{TOTP}}$. We assume that the challenger keeps a system clock T_{ct} which is updated every Δ_s. The goal adversary in the game is to forge a valid password of TOTP for a future time.

[6] Cryptographic schemes proven secure in the standard model, where adversaries are restricted only by time and computational power, are preferable. In this model, security analysis relies solely on the properties of the underlying (mathematical) assumptions, without relying on (idealized) random oracles. Canetti et al. [10] showed that some schemes secure in the random oracle model become insecure when used with specific hash functions.

$G^{\mathsf{TOTP_Forge}}_{\mathcal{A},\mathsf{TOTP}}(\kappa, T_s, T_e, \Delta_s)$:	
$\underline{\mathsf{Initialize}(T_s, T_e, \Delta_s)}$: $\mathsf{pms}_{\mathsf{TOTP}} \leftarrow \mathsf{TOTP.Setup}(1^\kappa, T_s, T_e, \Delta_s)$ $sd \xleftarrow{\$} \mathcal{K}_{\mathsf{TOTP}}$ $vp \leftarrow \mathsf{TOTP.PInit}(sd)$ OUTPUT $\mathsf{pms}_{\mathsf{TOTP}}, vp$	$\underline{\mathsf{Finalize}()}$: IF $\exists(pw^*, T^*) \in \mathsf{HD}$ s.t. ($\mathsf{TOTP.Verify}(vp, pw^*, T^*) = 1$ and no $\mathsf{GetNextPw}()$ at $\tilde{T}$ s.t. $\tilde{T} \geq T^*$ OUTPUT 1 OUTPUT 0
$\underline{\mathsf{GetNextPw}()}$: OUTPUT $\mathsf{TOTP.PGen}(sd, T_{ct})$	$\underline{\mathsf{ReceivePw}(pw)}$: APPEND $(pw, T_{ct}) \rightarrow \mathsf{HD}$ OUTPUT $\mathsf{TOTP.Verify}(vp, pw, T_{ct})$

Fig. 1: Procedures Used to Define Security for TOTP.

Definition 1. *We say a TOTP protocol is secure if no PPT adversary has a non-negligible advantage* $\mathsf{Adv}^{\mathsf{TOTP_Forge}}_{\mathcal{A},\mathsf{TOTP}}(\kappa, T_s, T_e, \Delta_s) := \Pr\left[G^{\mathsf{TOTP_Forge}}_{\mathcal{A},\mathsf{TOTP}}(\kappa, T_s, T_e, \Delta_s) = 1\right]$ *with given parameters.*

Pseudo-random Function Family. A pseudo-random function (PRF) family consists of two algorithms. $\mathsf{Setup}(1^\kappa)$ takes as input the security parameter 1^κ, and outputs a random secret key $k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$ and system parameters $\mathsf{pms}_{\mathsf{PRF}}$, where $\mathcal{K}_{\mathsf{PRF}}$ is the key space. $\mathsf{Eval}(k, x)$ takes as input the secret key k and a message $x \in \mathcal{M}_{\mathsf{PRF}}$, and outputs the evaluation result $r \in \mathcal{R}_{\mathsf{PRF}}$, where $\mathcal{M}_{\mathsf{PRF}}$ and $\mathcal{R}_{\mathsf{PRF}}$ are the message space and the range space, respectively. For a specific PRF function family F, we may write $\mathsf{F}(k, x)$ to represent $\mathsf{F.Eval}(k, x)$ for short. Let F be a pseudo-random function family associated with a key space $\mathcal{K}_{\mathsf{PRF}}$, a message space $\mathcal{M}_{\mathsf{PRF}}$, and a range space $\mathcal{R}_{\mathsf{PRF}}$. We define a security game $G^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}}(\kappa, q_f)$ (see also in Figure 2) that is played between a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ and a challenger based on F and the security parameter κ. During the experiment, the adversary is allowed to make the $\mathsf{Challenge}$ query only once and can issue at most q_f queries to FuncQ.

$G^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}}(\kappa, q_f)$:	
$\underline{\mathsf{Initialize}()}$: $(\mathsf{pms}_{\mathsf{PRF}}, k) \leftarrow \mathsf{F.Setup}(1^\kappa)$ OUTPUT $\mathsf{pms}_{\mathsf{PRF}}$	$\underline{\mathsf{Finalize}(b^*)}$: IF $b^* = b$ and $x^* \notin \mathsf{FL}$, OUTPUT 1 ELSE OUTPUT 0
$\underline{\mathsf{Challenge}(x^*)}$: $b \xleftarrow{\$} \{0,1\}$; $r_0 \xleftarrow{\$} \mathcal{R}_{\mathsf{PRF}}$; $r_1 \leftarrow \mathsf{F.Eval}(k, x^*)$ OUTPUT r_b	$\underline{\mathsf{FuncQ}(x)}$: APPEND $x \rightarrow \mathsf{FL}$ OUTPUT $\mathsf{F.Eval}(k, x)$

Fig. 2: Procedures Used to Define Security for PRF.

Definition 2. *We say* F *is secure if the advantage* $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}}(\kappa, q_f) := \left|\Pr[G^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}}(\kappa, q_f) = 1] - \frac{1}{2}\right|$ *of any PPT adversary* $\mathcal{A}$ *is negligible under* κ.

Collision-resistant Hash Functions. A collision-resistant hash function CRH is defined by the following two algorithms. $\mathsf{Setup}(1^\kappa)$ takes as input 1^κ, and outputs the parameter $\mathsf{pms}_{\mathsf{CRH}}$ and a random key $hk \xleftarrow{\$} \mathcal{K}_{\mathsf{CRH}}$, where $\mathcal{K}_{\mathsf{CRH}}$ is the hash key space. $\mathsf{Eval}(hk, m)$ takes as input a random key hk and a message $m \in \mathcal{M}_{\mathsf{CRH}}$, and outputs a hash value $y \in \mathcal{Y}_{\mathsf{CRH}}$, where $\mathcal{M}_{\mathsf{CRH}}$ is the message space and $\mathcal{Y}_{\mathsf{CRH}}$ is the hash value space. For a specific collision-resistant hash function H, we write $\mathsf{H}(m)$ to represent $\mathsf{H.Eval}(hk_{\mathsf{CRH}}, m)$ when hk_{CRH} is clear from the context. The CRH security game $G^{\mathsf{CR}}_{\mathcal{A},\mathsf{H}}(\kappa)$ based on an adversary $\mathcal{A}$ and a CRH family H is defined in Figure 3, where H has key space $\mathcal{K}_{\mathsf{CRH}}$, message space $\mathcal{M}_{\mathsf{CRH}}$ and hash value space $\mathcal{Y}_{\mathsf{CRH}}$.

$G^{\mathsf{CR}}_{\mathcal{A},\mathsf{H}}(\kappa)$:	
$\underline{\mathsf{Initialize}()}$: $(\mathsf{pms}_{\mathsf{CRH}}, hk) \leftarrow \mathsf{H.Setup}(1^\kappa)$ OUTPUT hk	$\underline{\mathsf{Finalize}(m, m')}$: IF $m \neq m'$ and $\mathsf{H.Eval}(hk, m) = \mathsf{H.Eval}(hk, m')$ OUTPUT 1 OUTPUT 0

Fig. 3: Procedures Used to Define Security for CRH.

Definition 3. *We say* H *is secure if no PPT adversary has non-negligible advantage* $\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\mathsf{H}}(\kappa) := \Pr[G^{\mathsf{CR}}_{\mathcal{A},\mathsf{H}}(\kappa) = 1]$ *under* κ.

Chameleon Hash Function. We define a chameleon hash function by the following three algorithms. $\mathsf{Setup}(1^\kappa; rk_{\mathsf{CH}})$ takes as input the security parameter 1^κ, and a randomness $rk_{\mathsf{CH}} \in \mathcal{R}_{\mathsf{CH}}$ for key generation, and outputs the system parameters $\mathsf{pms}_{\mathsf{CH}}$, and a pair of secret and public key $(sk_{\mathsf{CH}}, pk_{\mathsf{CH}}) \in \mathcal{K}_{\mathsf{CH}}$, where $\mathcal{K}_{\mathsf{CH}}$ is key space and $\mathcal{R}_{\mathsf{CH}}$ is randomness space. $\mathsf{Eval}(pk_{\mathsf{CH}}, m, r)$ is an evaluation algorithm that takes as input a public key $pk_{\mathsf{CH}} \in \mathcal{K}_{\mathsf{CH}}$, a message $m \in \mathcal{M}_{\mathsf{CH}}$ and a randomness $r \in \mathcal{R}_{\mathsf{CH}}$, and outputs a value $y \in \mathcal{Y}_{\mathsf{CH}}$, where $\mathcal{M}_{\mathsf{CH}}$ and $\mathcal{Y}_{\mathsf{CH}}$ are message space and hash value space, respectively. $\mathsf{Coll}(sk_{\mathsf{CH}}, m, r, m')$ is an efficient deterministic collision algorithm that takes as input the secret key sk_{CH} and tuple $(m, r, m') \in \mathcal{M}_{\mathsf{CH}} \times \mathcal{R}_{\mathsf{CH}} \times \mathcal{M}_{\mathsf{CH}}$, outputs a random value $r' \in \mathcal{R}_{\mathsf{CH}}$ such that $\mathsf{CH.Eval}(pk_{\mathsf{CH}}, m, r) = \mathsf{CH.Eval}(pk_{\mathsf{CH}}, m', r')$. We require that the secret key sk_{CH} can be efficiently generated by rk_{CH}. For example, for a discrete logarithm based CH, it is possible to set $sk_{\mathsf{CH}} := rk_{\mathsf{CH}}$. We define two standard security properties (i.e., collision resistance (CR) and uniformity (UN)) of a chameleon hash family CH in a single game $G^{\mathsf{CHS}}_{\mathcal{A},\mathsf{CH}}(\kappa)$ shown in Figure 4, where CH is associated with key space $\mathcal{K}_{\mathsf{CH}}$, randomness space $\mathcal{R}_{\mathsf{CH}}$, message space $\mathcal{M}_{\mathsf{CH}}$, and hash value space $\mathcal{Y}_{\mathsf{CH}}$.

$G^{\mathsf{CHS}}_{\mathcal{A},\mathsf{CH}}(\kappa)$:	
$\underline{\mathsf{Initialize}()}$:	$\underline{\mathsf{Finalize}(m, m', r_1, r_2)}$:
$rk \xleftarrow{\$} \mathcal{R}_{\mathsf{CH}}$	IF one of the following conditions is held:
$(\mathsf{pms}_{\mathsf{CH}}, pk, sk) \leftarrow \mathsf{CH.Setup}(1^\kappa; rk)$	i) $m \neq m'$ and $\mathsf{CH.Eval}(pk, m, r_1) = \mathsf{CH.Eval}(pk, m', r_2)$;
OUTPUT $\mathsf{pms}_{\mathsf{CH}}, pk$	ii) $r_1 \neq r_2$ and $\mathsf{CH.Eval}(pk, m, r_1) = \mathsf{CH.Eval}(pk, m, r_2)$;
	OUTPUT 1
	OUTPUT 0

Fig. 4: Procedures Used to Define Security for CH.

Definition 4. *We say* CH *is secure if no PPT adversary has non-negligible advantage* $\mathsf{Adv}^{\mathsf{CHS}}_{\mathcal{A},\mathsf{CH}}(\kappa) := \Pr[G^{\mathsf{CHS}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1]$ *under* κ.

Merkle Tree. We define a Merkle tree scheme with three algorithms. $\mathsf{Build}(\{\mathrm{lf}_i\}_{i\in[\ell]})$ takes as input ℓ data items (which form the leaves of the target tree) and builds a Merkle tree instance MTI on top of them. We will reuse the notion of MTI to denote the root of the Merkle tree instance as well. Let H be a collision-resistant hash function. Specifically, each non-leaf node is a hash value $\mathsf{H}(\text{node.LeftChild}||\text{node.RightChild})$. $\mathsf{GetPrf}(\mathrm{MTI}, \mathrm{lf}_i)$ takes as input a Merkle tree MTI and a leaf lf_i, and outputs a proof $\mathrm{Pf}_{\mathrm{lf}_i}$ that can attest the inclusion of lf_i in the tree. The proof $\mathrm{Pf}_{\mathrm{lf}_i}$ encompasses the siblings of every node on the path from lf_i to the tree's root MTI. $\mathsf{Verify}(\mathrm{MTI}, \mathrm{lf}_i, \mathrm{Pf}_{\mathrm{lf}_i})$ takes as input the root MTI, a leaf node lf_i, and the corresponding proof $\mathrm{Pf}_{\mathrm{lf}_i}$. It computes a root MTI based on lf_i and proof $\mathrm{Pf}_{\mathrm{lf}_i}$, and returns 1 if MTI = MTI and 0 otherwise. For a secure Merkle tree scheme, the adversary must not forge the Merkle proof for a leaf node that does not belong to the Merkle tree. We define a security game $G^{\mathsf{MT_Forge}}_{\mathcal{A},\mathsf{MT}}(\kappa)$ in Figure 5 for a Merkle tree scheme MT, which encompasses the following procedures. For a secure Merkle tree scheme, the adversary must not forge the Merkle proof for a leaf node which does not belong to the Merkle tree.

$G^{\mathsf{MT_Forge}}_{\mathcal{A},\mathsf{MT}}(\kappa)$:	
$\underline{\mathsf{Initialize}(\{\mathrm{lf}_i\}_{i\in[\ell]})}$:	$\underline{\mathsf{Finalize}(\mathrm{lf}^*, \mathrm{Pf}^*)}$:
$\mathrm{MTI} \leftarrow \mathsf{MT.Build}(\{\mathrm{lf}_i\}_{i\in[\ell]})$	IF $1 \leftarrow \mathsf{MT.Verify}(\mathrm{Rt}, \mathrm{lf}^*, \mathrm{Pf}^*)$
OUTPUT MTI	and $\mathrm{lf}^* \notin \{\mathrm{lf}_i\}_{i\in[\ell]}$, OUTPUT 1
	OUTPUT 0

Fig. 5: Procedures Used to Define Security for MT.

Definition 5. *We say* MT *is secure if no PPT adversary has non-negligible advantage* $\mathsf{Adv}^{\mathsf{MT_Forge}}_{\mathcal{A},\mathsf{MT}}(\kappa) := \Pr[G^{\mathsf{MT_Forge}}_{\mathcal{A},\mathsf{MT}}(\kappa) = 1]$ *under* κ.

Unpredictable Permutation. We consider a keyed permutation scheme with two algorithms. $\mathsf{Setup}(1^\kappa)$ takes as input the security parameter 1^κ, and outputs a random permutation key $k \xleftarrow{\$} \mathcal{K}_{\mathsf{PM}}$ and other the system parameters $\mathsf{pms}_{\mathsf{PM}}$, where $\mathcal{K}_{\mathsf{PM}}$ is a key space. $\mathsf{Shuffle}(k, X)$ takes as input a permutation key k and a set of elements $X = \{x_i\}_{i\in[n]} \in \mathcal{M}_{\mathsf{PM}}$ of size $n \in \mathbb{N}$, and outputs a permuted set $Y \in \mathcal{M}_{\mathsf{PM}}$, where $\mathcal{M}_{\mathsf{PM}}$ is an element space. In Figure 6, we define a security game $G^{\mathsf{UP}}_{\mathcal{A},\mathsf{PM}}(\kappa)$ that is played between a PPT adversary $\mathcal{A}$ and a challenger based on a permutation scheme PM and the security parameter κ, where PM has the key space $\mathcal{K}_{\mathsf{PM}}$ and the element space $\mathcal{M}_{\mathsf{PM}}$.

Definition 6. *We say* PM *is secure if for any PPT adversaries, the advantage* $\mathsf{Adv}^{\mathsf{UP}}_{\mathcal{A},\mathsf{PM}}(\kappa) := \left|\Pr[G^{\mathsf{UP}}_{\mathcal{A},\mathsf{PM}}(\kappa) = 1] - \frac{1}{2}\right|$ *is negligible under* κ.

$G^{\mathsf{UP}}_{\mathcal{A},\mathsf{PM}}(\kappa)$:	
$\underline{\mathsf{Initialize}(n)}$:	$\underline{\mathsf{Finalize}(\beta^*)}$:
$(\mathsf{pms}_{\mathsf{PM}}, k) \leftarrow \mathsf{PM.Setup}(1^\kappa, n)$	IF $\beta^* = \beta_0$, OUTPUT 1
OUTPUT $\mathsf{pms}_{\mathsf{MT}}$	OUTPUT 0
$\underline{\mathsf{Challenge}(\{x_i\}_{i\in[n]}, w_0, w_1)}$:	
IF $w_0, w_1 \notin [n]$ or $w_0 = w_1$ OUTPUT $\perp$	
$\{y_i\}_{i\in[n]} := \mathsf{PM.Shuffle}(k, \{x_i\}_{i\in[n]})$	
Let β_0 and β_1 be indices s.t. $y_{\beta_0} = x_{w_0}$ and $y_{\beta_1} = x_{w_1}$	
OUTPUT $\{y_i\}_{i\in[n\setminus\{\beta_0,\beta_1\}]}$	

Fig. 6: Procedures Used to Define Security for PM.

Authenticated Symmetric Encryption. An authenticated symmetric encryption scheme ASE has three algorithms. $\mathsf{Setup}(1^\kappa)$ takes as input the security parameter 1^κ, and outputs the parameter $\mathsf{pms}_{\mathsf{ASE}}$ and a random key $k \xleftarrow{\$} \mathcal{K}_{\mathsf{ASE}}$, where $\mathcal{K}_{\mathsf{ASE}}$ is the key space. $\mathsf{Enc}(k, m; re)$ takes as input an encryption key k, a message $m \in \mathcal{M}_{\mathsf{ASE}}$, and a random value $re \in \mathcal{R}_{\mathsf{ASE}}$, and outputs a ciphertext $C \in \mathcal{C}_{\mathsf{ASE}}$, where $\mathcal{M}_{\mathsf{ASE}}$ is the message space, $\mathcal{C}_{\mathsf{ASE}}$ is the ciphertext space, and $\mathcal{R}_{\mathsf{ASE}}$ is the randomness space. Due to the randomness r, the encryption algorithm Enc is a probabilistic (randomized) algorithm. $\mathsf{Dec}(k, C)$ takes as input an encryption key k, and a ciphertext $C \in \mathcal{C}_{\mathsf{ASE}}$, and outputs a message $m \in \mathcal{M}_{\mathsf{ASE}}$. Let ASE be an authenticated symmetric encryption scheme associated with key space $\mathcal{K}_{\mathsf{ASE}}$, message space $\mathcal{M}_{\mathsf{ASE}}$, ciphertext space $\mathcal{C}_{\mathsf{ASE}}$, and randomness space $\mathcal{R}_{\mathsf{ASE}}$. We define a security game $G^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{ASE}}(\kappa, q_e)$ in Figure 7 to formulate the standard notion of indistinguishability under chosen-ciphertext attacks (IND-CCA), that is played between a PPT adversary $\mathcal{A}$ and a challenger based on ASE and κ. Meanwhile, the Challenge query can be made at most once, and the messages involved must have equal lengths within the message space. Additionally, the DecP and the EncP = procedures can be queried at most q_e times, respectively.

$G^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{ASE}}(\kappa, q_e)$:		
$\underline{\mathsf{Initialize}()}$:	$\underline{\mathsf{Finalize}(b^*)}$:	$\underline{\mathsf{Challenge}(m_0, m_1)}$:
$(\mathsf{pms}, k) \leftarrow \mathsf{ASE.Setup}(1^\kappa)$	IF $b^* = b$ and $C^* \notin \mathsf{CL}$	$b \xleftarrow{\$} \{0,1\}$
OUTPUT pms	OUTPUT 1	$C^* \leftarrow \mathsf{ASE.Enc}(k, m_b)$
	OUTPUT 0	OUTPUT C^*
$\underline{\mathsf{DecP}(C)}$:	$\underline{\mathsf{EncP}(m)}$:	
APPEND $C \rightarrow \mathsf{CL}$	$re \xleftarrow{\$} \mathcal{R}_{\mathsf{ASE}}$	
OUTPUT $\mathsf{ASE.Dec}(k, C)$	OUTPUT $\mathsf{ASE.Enc}(k, m; re)$	

Fig. 7: Procedures Used to Define Security for ASE.

Definition 7. *We say* ASE *is secure if for any PPT adversaries, the advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{ASE}}(\kappa, q_e) := \left|\Pr[G^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{ASE}}(\kappa, q_e) = 1] - \frac{1}{2}\right|$ *is negligible under* κ.

3 Security Notions of Dynamic Group Time-based One-time Passwords

This section defines the security notions of DGTOTP.

Entities. In a DGTOTP scheme, there are three kinds of entities encompassing group members, verifiers, and an honest registration authority (RA). Each group member has a unique identity ID. RA is trustworthy and takes responsibility for initializing the system and enrolling group members.

Syntax. We define a DGTOTP scheme via the following eight algorithms and a sub-protocol.

- $(\mathsf{pms}, \mathsf{gpk}_{\mathsf{G}}, \mathsf{GM}_{\mathsf{G}}, \mathrm{RL}_{\mathsf{G}}, sk_{\mathsf{RA}}) \leftarrow \mathsf{RASetup}(1^\kappa, \mathsf{setpms})$: RA runs the system setup algorithm to initialize the system based on the security parameter and the setup parameters $\mathsf{setpms} = (\mathsf{G}, U, T_s, T_e, \Delta_e, \Delta_s)$. This algorithm outputs the system parameters pms, the initial group public key $\mathsf{gpk}_{\mathsf{G}}$ of a group G, the corresponding initial group management message GM_{G}, a revocation list RL_{G}, and a secret key $sk_{\mathsf{RA}} \xleftarrow{\$} \mathcal{K}_{\mathsf{RA}}$ for RA, where $\mathcal{K}_{\mathsf{RA}}$ is a key space of RA, the parameter $U \in \mathbb{N}$ specifies the maximum number of group members that a group can hold, T_s and T_e are the start and the end time of a DGTOTP protocol instance, Δ_s is an interval between two passwords, and Δ_e is a parameter to define the verification epoch of a verification point of a TOTP instance. The group management message GM_{G} is stored privately by RA.
- $(sk_{\mathsf{ID}_j}, \mathsf{vst}_{\mathsf{ID}_j}) \leftarrow \mathsf{PInit}(\mathsf{ID}_j)$: A group member ID_j runs the member initialization algorithm to generate its secret key $sk_{\mathsf{ID}_j} \xleftarrow{\$} \mathcal{K}_{\mathsf{ID}}$ and the initial verification state $\mathsf{vst}_{\mathsf{ID}_j}$, where $\mathcal{K}_{\mathsf{ID}}$ is a key space of the group members.

$G^{\mathsf{Gvar}}_{\mathcal{A},\Sigma}(\kappa, \mathsf{setpms})$:

Initialize() :
$(\mathsf{pms}, \mathsf{gpk_G}, \mathsf{GM_G}, \mathrm{RL_G}, sk_{\mathsf{RA}}) \leftarrow \Sigma.\mathsf{RASetup}(1^\kappa, \mathsf{setpms})$
Create lists $(\mathrm{CML}, \mathrm{CSL}, \mathrm{HML}) \leftarrow \emptyset$
Init a monotonically increasing (in Δ_s) system clock $T_{ct} := T_s$
OUTPUT $\mathsf{pms}, \mathsf{gpk_G}, \mathrm{RL_G}$

Challenge$(\mathsf{ID}_0, \mathsf{ID}_1)$:
IF $(\hat{\mathsf{ID}_0}, \hat{\mathsf{ID}_1}) \notin \mathrm{HML}$ or $\hat{\mathsf{ID}_0} = \hat{\mathsf{ID}_1}$
  OUTPUT $\perp$
$b \xleftarrow{\$} \{0,1\}$
Switch to the next verification epoch
$sd_{\hat{\mathsf{ID}_b}} \leftarrow \Sigma.\mathsf{GetSD}(sk_{\hat{\mathsf{ID}_b}}, T_{ct})$
Switch to the next verification epoch
OUTPUT $sd_{\hat{\mathsf{ID}_b}}$

Corrupt(ID_j) :
IF $\mathsf{ID}_j \notin \mathrm{HML}$, OUTPUT $\perp$
$\mathrm{CML} \leftarrow \mathsf{ID}_j$
OUTPUT sk_{ID_j}

CompromiseSD(ID_j) :
IF $\mathsf{ID}_j \notin \mathrm{HML}$, OUTPUT $\perp$
$sd^i_{\mathsf{ID}_j} \leftarrow \Sigma.\mathsf{GetSD}(sk_{\mathsf{ID}_j}, T_{ct})$
$\mathrm{CSL} \leftarrow sd^i_{\mathsf{ID}_j}$
OUTPUT $sd^i_{\mathsf{ID}_j}$

GetNextPw() :
FOR $\forall \mathsf{ID}_j \in \mathrm{HML}$:
  $sd^i_{\mathsf{ID}_j} \leftarrow \Sigma.\mathsf{GetSD}(sk_{\mathsf{ID}_j}, T_{ct})$
  $pw_{\mathsf{ID}_j} \leftarrow \Sigma.\mathsf{PwGen}(sd^i_{\mathsf{ID}_j}, T_{ct})$
$\mathrm{PL} \leftarrow (\{pw_{\mathsf{ID}_j}\}_{\mathsf{ID}_j \in \mathrm{HML}}, T_{ct})$
OUTPUT $\{pw_{\mathsf{ID}_j}\}_{\mathsf{ID}_j \in \mathrm{HML}}$

Finalize(b^*, pw^*, T^*) :
$\mathsf{ID}^* := \Sigma.\mathsf{Open}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, pw^*, T^*)$
$vr := \Sigma.\mathsf{Verify}(\mathsf{gpk_G}, pw^*, T^*, \mathrm{RL_G})$
$i := \frac{\lceil T^* - T_s \rceil}{\Delta_e}$; $\tilde{T}^* := T^* - ((T^* - T_s) \mod \Delta_s)$
IF Gvar = Corr and $pw^* \in \mathrm{PL}$ and $(vr = 0 \vee \mathsf{ID}^* \notin \mathrm{HML})$
  OUTPUT 1
IF Gvar = Anony
  IF $\hat{\mathsf{ID}_0} \in (\mathrm{CML} \cup \mathrm{OL})$ or $\hat{\mathsf{ID}_1} \in (\mathrm{CML} \cup \mathrm{OL})$, OUTPUT $\perp$
  IF $b = b^*$, OUTPUT 1
IF Gvar = Trace and $vr = 1$
  and $\Big((\mathsf{ID}^* \notin (\mathrm{HML} \cup \mathrm{CML})) \ \vee \ ((pw^*, \tilde{T}^*) \notin \mathrm{PL} \ \wedge \ \mathsf{ID}^* \notin \mathrm{CML} \ \wedge \ sd^i_{\mathsf{ID}^*} \notin \mathrm{CSL})\Big)$
  OUTPUT 1
OUTPUT 0

AddHM(ID_j) :
$(sk_{\mathsf{ID}_j}, \mathsf{vst}_{\mathsf{ID}_j}) \leftarrow \mathsf{PInit}(\mathsf{ID}_j)$
$(\mathsf{Ax}_{\mathsf{ID}_j}, \mathsf{gpk'_G}, \mathsf{GM'_G}) \leftarrow \Sigma.\mathsf{Join}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, \mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$
$\mathrm{HML} \leftarrow \mathsf{ID}_j$
OUTPUT $\mathsf{Ax}_{\mathsf{ID}_j}, \mathsf{gpk'_G}, \mathsf{GM'_G}[\mathsf{ID}_j]$

AddMM$(\mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$:
$(\mathsf{Ax}_{\mathsf{ID}_j}, \mathsf{gpk'_G}, \mathsf{GM'_G}) \leftarrow \Sigma.\mathsf{Join}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, \mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$
$\mathrm{CML} \leftarrow \mathsf{ID}_j$
OUTPUT $\mathsf{Ax}_{\mathsf{ID}_j}, \mathsf{gpk'_G}, \mathsf{GM'_G}[\mathsf{ID}_j]$

ReceivePw(pw) :
OUTPUT $\Sigma.\mathsf{Verify}(\mathsf{gpk_G}, pw, T_{ct}, \mathrm{RL_G})$

OpenID(pw, T) :
$\mathsf{ID} := \Sigma.\mathsf{Open}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, pw, T)$, $\mathrm{OL} \leftarrow \mathsf{ID}$
OUTPUT ID

RevokeID(ID_j) :
OUTPUT $\mathrm{RL'_G} \leftarrow \Sigma.\mathsf{Revoke}(sk_{\mathsf{RA}}, \mathsf{ID}_j, \mathsf{gpk_G}, \mathrm{RL_G}, \mathsf{GM_G})$

Fig. 8: Procedures Used to Define the Security of a DGTOTP Scheme.

- $(\mathsf{Ax}_{\mathsf{ID}_j}, \mathsf{gpk'_G}, \mathsf{GM'_G}) \leftarrow \mathsf{Join}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, \mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$: RA runs this join algorithm to handle the enrolment of the group member ID_j (who provides with its verification sate $\mathsf{vst}_{\mathsf{ID}_j}$) to the group with public key $\mathsf{gpk_G}$. It returns the auxiliary information $\mathsf{Ax}_{\mathsf{ID}_j}$ as registration receipt to ID_j, and updates the group management message $\mathsf{GM'_G}$ and the group public key $\mathsf{gpk'_G}$ (if necessary).
- $sd^i_{\mathsf{ID}_j} \leftarrow \mathsf{GetSD}(sk_{\mathsf{ID}_j}, T)$: A group member ID_j runs this seed generation algorithm using its secret key sk_{ID_j} to compute the secret seed $sd^i_{\mathsf{ID}_j} \in \mathcal{S}_{\mathsf{DGTOTP}}$ for generating the password at T, where $\mathcal{S}_{\mathsf{DGTOTP}}$ is a secret seed space.
- $pw^{i,z}_{\mathsf{ID}_j} \leftarrow \mathsf{PwGen}(sd^i_{\mathsf{ID}_j}, T)$: A group member ID_j runs the password generation algorithm using its secret seed $sd^i_{\mathsf{ID}_j}$ for the time slot T to generate the corresponding one-time password $pw^{i,z}_{\mathsf{ID}_j}$, where z is an index of the password in the i-th verification epoch defined by T.
- $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{gpk_G}, pw^{i,z}_{\mathsf{ID}_j}, T, \mathrm{RL_G})$: A verifier runs the password verification algorithm based on the group public key $\mathsf{gpk_G}$ and the revocation list $\mathrm{RL_G}$ to verify the the password $pw^{i,z}_{\mathsf{ID}_j}$ for the time slot T. This algorithm outputs 1 if $pw^{i,z}_{\mathsf{ID}_j}$ is accepted, and 0 otherwise.
- $\mathrm{RL'_G} \leftarrow \mathsf{Revoke}(sk_{\mathsf{RA}}, \mathsf{ID}_j, \mathsf{gpk_G}, \mathrm{RL_G}, \mathsf{GM_G})$: RA runs the revocation algorithm using its secret key sk_{RA}, the revoking identity ID_j, and the group management message $\mathsf{GM_G}$, to revoke the credentials of ID_j with respect to $\mathsf{gpk_G}$ and update the revocation list to $\mathrm{RL'_G}$.
- $\mathsf{ID}_j \leftarrow \mathsf{Open}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, pw^{i,z}_{\mathsf{ID}_j}, T)$: RA runs the identity extraction algorithm using its secret key sk_{RA} to extract the identity ID_j of the owner of a password $pw^{i,z}_{\mathsf{ID}_j}$ w.r.t. the time slot T and group public key $\mathsf{gpk_G}$. The algorithm outputs ID_j if the extraction procedure is successful and $\perp$ otherwise.

Threat Model. We introduce the threats against DGTOTP in terms of the entities in the system following the similar settings in literature (e.g., [14,39,33,52,23]). We assume a system that exists at least two honest group members. In other words, most of the group members can be malicious and controlled by attackers. Those malicious group members can be seen as insider threats against the privacy of other honest group members. In the dynamic group management setting, an attacker may adaptively register new malicious group members with arbitrary identities at any time. The verifier can also be malicious. The malicious group members and verifiers may infer the private information of honest group members. Meanwhile, we consider RA to be a semi-honest and non-colluding third party. RA might faithfully handle the enrolment of group members and trace the identity of malicious group members but may be curious about how (where and when) those passwords are used. We assume the attackers take control of the network traffic, so they can intercept, inject, and manipulate the communication. In addition, the attackers may also try to either impersonate honest group members (i.e., breaking the *traceability* of DGTOTP) by forging their unused passwords or creating passwords that cannot

be traced by RA.[7] Like pseudonym schemes (e.g., [12,13,37,50]), attackers are interested in the privacy of an honest group member's passwords belonging to different verification epochs. That is, an attacker, who knows all identities in a group, may try to link a password to the real identity of its owner (i.e., breaking the *anonymity* of DGTOTP). However, we do not consider the privacy leakage from the usage pattern of passwords (in one verification epoch). We present more discussions on the privacy issue in Section 8.

Security Definition. As DGTOTP is a lightweight cryptographic primitive, we here consider the most desirable security properties we can achieve without introducing much performance overhead. We define the correctness (Corr) and security properties of DGTOTP within a unified game-based framework [5]. The security properties that we consider include the *anonymity* (Anony), and *traceability* (Trace) [8]. Each property is associated with a game indexed by a variable $\mathsf{Gvar} \in \{\mathsf{Corr}, \mathsf{Anony}, \mathsf{Trace}\}$. These games consist of a series of procedures, as shown in Figure 8, which are simulated by a challenger. An adversary can start a game by calling Initialize and end the game with Finalize. All other procedures can be queried sequentially and adaptively by the adversary. The adversary seeks to achieve its capabilities via given queries to meet the winning conditions (defined in Finalize) of the corresponding game, i.e., resulting in $\mathsf{Finalize}(b^*, pw^*, T^*) = 1$.

The anonymity and the traceability are adapted from [56] but under a dynamic group member setting. To generically model the attacks against join operations (e.g., malicious registration attacks and adaptive chosen identity attacks), we allow an adversary to adaptively add (at any time of the game) either honest group members via the AddHM procedure or malicious group members via the AddMM procedure, respectively. Hence, an adversary can control those maliciously registered parties, which also covers insider threats. For a malicious group member ID_j, the adversary generates the verification state vst_{ID_j}, so the corresponding secrets of ID_j are not known to the challenger. The adversary can reveal an honest member's long-term secret key and secret seeds via Corrupt and CompromiseSD queries, respectively. This models the real-world compromise of the credentials of an honest party (e.g., key stolen by insiders, poor key management, and cryptanalysis). The password generation and verification procedures of honest group members are simulated by the GetNextPw and ReceivePw procedures, respectively. The GetNextPw query is used to model the known password attacks that require the leaked passwords of honest parties not to affect the security properties of their unexposed passwords. Meanwhile, the ReceivePw procedure allows an adversary to test her forged passwords, exemplifying scenarios such as impersonation attacks. The adversary can also revoke a group member anytime via the RevokeID procedure. This procedure (working together with AddMM and AddHM procedures) can be used to model the malicious capabilities of an adversary in manipulating the group organization structure and then inferring the privacy information of other honest group members. In addition, an adversary can ask the OpenID procedure to disclose the identities of passwords belonging to honest group members. For privacy, it is imperative that the revelation of one password does not compromise the other unrevealed passwords.

Meanwhile, the anonymity requires that the adversary cannot distinguish which two honest members' secret seeds in the Challenge procedure. In addition, we impose the restriction that the Challenge procedure can be invoked at most once during the game. The traceability requires that the adversary cannot generate a valid password-time pair (pw^*, T^*) that meets one of the following conditions: i) it is opened to an identity (including $\perp$) which is not involved in any AddHM and AddMM queries; ii) it belongs to an uncorrupted honest party (either its long-term key or secret seed for generating pw^* is not exposed) and is not generated by the challenger in any procedure.

Definition 8. *We say that a DGTOTP scheme Σ is correct if the probability* $\Pr\left[G^{\mathsf{Corr}}_{\mathcal{A},\Sigma}(\mathsf{Gpm}) = 0\right] \approx 1$ *holds for any PPT adversary $\mathcal{A}$ and parameters* $\mathsf{Gpm} = (\kappa, \mathsf{setpms})$ *and* $\mathsf{setpms} = (\mathsf{G}, U, T_s, T_e, \Delta_e, \Delta_s)$. *We say that a correct Σ (with the above parameters) is secure if the advantages* $\mathsf{Adv}^{\mathsf{Anony}}_{\mathcal{A},\Sigma}(\mathsf{Gpm}) := \left|\Pr\left[G^{\mathsf{Anony}}_{\mathcal{A},\Sigma}(\mathsf{Gpm}) = 1\right] - 1/2\right|$ *and* $\mathsf{Adv}^{\mathsf{Trace}}_{\mathcal{A},\Sigma}(\mathsf{Gpm}) := \Pr\left[G^{\mathsf{Trace}}_{\mathcal{A},\Sigma}(\mathsf{Gpm}) = 1\right]$ *of any PPT $\mathcal{A}$ are negligible in the corresponding games.*

Model Comparison. Our DGTOTP security model is strengthened and adapted from the static GTOTP security model [56] though they have similar security properties. In Table 1, we summarize the differences between these two models. Specifically, the GTOTP security model only formulates the selectively chosen identity attacks. All identities should be chosen by the adversary (without being capable of choosing their secret keys) at the beginning of the security game, and the adversary cannot revoke them. As a result, it cannot model the group structure manipulation capabilities of the adversary as in our new DGTOTP model. Contrastingly, our new model provides the AddHM, AddMM, and RevokeID procedures, allowing stronger adversaries to adaptively manipulate the group structure at any time throughout the security game, including chosen identities and credentials of malicious group members. In particular, the AddMM and the RevokeID queries empower adversaries

[7] Namely, the traceability of DGTOTP, like group signatures [4,8], is formulated to cover both unforgeability and accountability. Hence, traceability can deter malicious behavior within the group, thus mitigating insider threats (as individuals may be less likely to engage in malicious activities if they know their actions can be traced by RA).

[8] According to the two security properties, the adversaries can be divided into two types on breaking them respectively, i.e., anonymity adversaries and traceability adversaries.

to plant malicious insiders adaptively. Essentially, these new procedures strengthen the security properties of anonymity and traceability of DGTOTP compared to the GTOTP security model.

Table 1: Model Comparison

Model	Adversarial Capabilities				
	Differences				Common Procedures
	AddMM	AddHM	RevokeID	Init	Finalize, Challenge, OpenID GetNextPw, ReceivePw Corrupt, CompromiseSD
GTOTP	×	×	×	Keys & Parameters & Group Members	
DGTOTP	√	√	√	Keys & Parameters	

4 An Efficient DGTOTP Scheme

This section presents an efficient dynamic group time-based one-time passwords scheme DGTOne.

4.1 Construction Overview

Design Goals. There are four major design goals (requirements) that we aim to achieve in DGTOne:
- **G1: Efficient dynamic group management**. Since it might be inconvenient for group members to update the group public key whenever the group is changed, we want to design a dynamic group management solution without disrupting the exiting group members (i.e., without changing the group public key and the local states of existing group members) while keeping the lightweight feature of TOTP.
- **G2: Constant storage overhead at group members**. It is always important to reduce storage costs for resource-constrained devices, saving for critical applications.
- **G3: Usage privacy of passwords against** RA. Since RA is semi-honest by assumption, it should not know the usage status (e.g., usage time and verifier) of passwords from the protocol messages it sent and received, unless RA is explicitly requested by a trustworthy authority to trace the identity of a password.
- **G4: Standard model security**. We aim to build a provable secure DGTOTP scheme in our DGTOTP security model without random oracles.

Construction Problems and Challenges. A fundamental construction problem is how to generate unlinkable group membership proofs of passwords in different verification epochs with the above design goals. In the meantime, the lightweight requirement of DGTOTP would exclude many public key cryptographic primitives (such as zero-knowledge proof systems, and oblivious pseudo-random functions).

To achieve anonymity, we stick to the traditional idea of pseudonym schemes, which is efficient to implement and allows a modular design of DGTOTP. Namely, we intend to generically transform a TOTP scheme into a DGTOTP scheme. In this way, each party can generate many TOTP verification points, each of which is used for verifying passwords in one verification epoch. A key problem regarding the design of unlinkable group membership proofs lies in devising a randomization solution capable of eliminating the linkability among the group membership proofs of verification points. It is a challenging task since all group members can dynamically join and leave the group with random verification points (which are unpredictable). Hence, the customized randomization scheme must remain independent of the post-determined verification points.[9] The randomization solution may significantly impact the storage cost of either RA or group members. For a group with U group members and E verification epochs, $O(U \cdot E)$ group membership proofs need to be randomized. A naive randomization procedure based on symmetric-key based techniques (e.g., permutation) may incur a considerable storage cost concerning the randomization results (e.g., $O(E)$ new positions in a permuted set for a group member), and therefore violates the design goal G2. One of our goals is to design a storage-friendly randomization solution. In addition, another challenge arises in designing an authentication scheme that can adaptively bind a verification point to a randomized group membership proof. Because of the requirement of **G3**, we cannot rely on RA (as in [58,57]) to online authenticate information (e.g., verification points) for group members during the usage of passwords. Meanwhile, all construction details should satisfy **G4**, which makes it harder to build, especially when considering adversaries who can adaptively register both honest and malicious group members and compromise honest group members.

Design Ideas. In order to avoid using public cryptographic operations for group members, we will leverage the hash-based Merkle tree scheme to generate group membership proofs. Namely, each verification point is supposed to be linked with a Merkle proof somehow (which will be detailed later). However, we cannot directly

[9] For example, we can no longer use the randomization technique in the static GTOTP scheme (i.e., YJN+), which permutes all verification points of group members in advance.

build the Merkle tree based on permuted verification points (as in the YJN+ scheme), since they are not predetermined in the dynamic group management scenario. In addition, we also want to avoid the high storage cost of the YJN+ scheme on the side of each group member.

Hence, our construction starts from figuring out a new Merkle tree organization approach that can fit our design goals. We observe that the verification epochs of all verification points are public information that cannot be hidden by the randomization operations. In other words, it is only necessary to randomize the order of the verification points of different group members in each verification epoch. Therefore, we can organize the leaves (i.e., associated with verification points) of the Merkle tree in terms of verification epochs. Namely, we can split the Merkle tree into E sub-Merkle trees, each of which contains the verification points belonging to the same verification epoch. The roots of sub-Merkle trees can be used as leaves to create the final Merkle tree, whose root is taken as the group public key. In this way, all parties will share the roots of sub-Merkle trees, which can be stored by any parties (incl. group members and RA) without any privacy leakage. In particular, a segment of a Merkle proof for the path from the root of the corresponding sub-Merkle tree to the group public key can be generated by the parties who have the roots of sub-Merkle trees. This fact can reduce Merkle proofs' storage. In a nutshell, we stress that the new Merkle tree organization approach lays down the foundation of our other construction gadgets (in particular for our randomization and outsourcing solutions, and security reduction).

For realizing **G1** and **G4**, we implement an issue-first-and-join-later (IFJL) strategy by leveraging chameleon hash function (CH) [40] to handle the join operations. Specifically, RA can compute all Merkle tree leaves $\{h_j^i := \mathsf{CH.Eval}(pk_j^i, dvp_j^i, rd_j^i)\}_{i\in[E],j\in[U]}$ based on $U \cdot E$ dummy verification point and randomness pairs. Note that each leaf is associated with a fresh chameleon hash key pair (sk_j^i, pk_j^i), so that we can faithfully answer the AddMM queries to correctly generate the collisions generation for maliciously chosen verification points (i.e., resilient to malicious registration attacks). Then, RA builds a Merkle tree using $\{h_j^i\}_{i\in[E],j\in[U]}$, and the resulting root is the group public key. We particularly bind each leaf with a chameleon hash key pair, which will be used for generating the chameleon collision for a real verification point of a group member. However, an open problem is how to randomly map a dynamically joined identity to achieve anonymity. Notably, RA cannot *directly* link an identity (that could be an arbitrary string) to a randomly allocated Merkle tree leaf in advance. To overcome this obstacle, we first map the joined identities to a sorted set of numbers in $[U]$ (e.g., according to their joined time). RA can run a timestamp service to sequentially generate timestamps for the joined group members, ensuring each group member can get a unique joined time stamp. It's worth mentioning that all other join steps can be run in parallel. The benefit of such an identity-transforming strategy enables us to further indirectly randomly map. Recall that each sub-Merkle tree (for a verification epoch) has U leaves, and each group member can have at most one leaf in that sub tree. Since the transformed identities and the leaves in each sub-Merkle tree have a bijective relation, we can apply a permutation scheme with verify-epoch dependent keys to create random one-to-one connections among them in each verification epoch to realize anonymity with **G4** (e.g., using an independent permutation key to simulate the challenge query).

To fully implement the above IFJL strategy and **G2** under the restriction of **G3**, we also need a solution that can generate chameleon collisions to bind a verification point with a Merkle tree leaf on the fly while preserving the usage privacy against RA. We first observe that the requirement **G3** would certainly rule out the chameleon-hash adaption at RA. To realize both **G2** and **G3**, we let RA reversely outsource chameleon-collision generation capability to group members themselves. I.e., RA generates independent chameleon secret keys to group members in the join procedure. This can relieve the heavy burdens of group members from storing the chameleon collisions. Our outsourcing solution is bilateral. Namely, RA should also help group members to generate chameleon public keys and Merkle proofs for verification. For efficiency, RA can compute them epoch by epoch. That is, RA only needs to compute and publish the chameleon public keys and the necessary ingredients for generating the Merkle proofs used in the current active verification epoch. A direct benefit of our whole IFJL strategy is that it can free a group member from generating all verification points in the initialization algorithm PInit, thereby reducing the memory overhead for the group member. Besides, our IFJL strategy also enables an optimal revocation feature since RA can simply terminate the outsourcing service of a revoked group member.

4.2 Detailed Algorithms

The algorithms of DGTOne are described as follows:

– $\mathsf{RASetup}(1^\kappa, \mathsf{setpms})$: RA defines parameters $\mathsf{setpms} = (U, T_s, T_e, \Delta_e, \Delta_s)$ for a group instance G according to the security parameter κ, where T_s is particularly set to the current time T_{ct}, and U is even (for sub-Merkle-tree generation). Next, RA initializes one PRF as $(\mathsf{pms}_{\mathsf{F}_1}, k_{\mathsf{RA}}) \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\kappa)$ (if its has not been initialized before). We assume that the ranges of the PRFs used in the following algorithms match the corresponding spaces of other cryptographic building blocks (e.g., the key space of ASE and the randomness space of CH). Then, RA initializes a collision-resistant hash function $(\mathsf{pms}_{\mathsf{CRH}}, hk) \leftarrow \mathsf{H}_1.\mathsf{Setup}(1^\kappa)$. For TOTP instances, RA sets the parameter $E := (T_e - T_s)/\Delta_e$, where E is the number of TOTP protocol instances to be used. It sets the system parameters $\mathsf{pms} := (\mathsf{pms}_{\mathsf{F}_1}, \mathsf{pms}_{\mathsf{F}_1}, \mathsf{setpms})$ and secret key $sk_{\mathsf{RA}} := k_{\mathsf{RA}}$. Subsequently, RA initializes the group public key $\mathsf{gpk}_{\mathsf{G}}$ through the following steps:

- Generate E sets of dummy verification points $\{dvp_j^i\}_{j\in[U]}$ for U group members, where $dvp_j^i := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“DVP”}||i)$ and $ks_j := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||j)$;
- Generate E sets of chameleon hash keys $\{pk_j^i\}_{j\in[U]}$, where $pk_j^i := \mathsf{CH.Setup}(1^\kappa; rk_j^i)$ and $rk_j^i := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“CHR”}||i)$;
- Compute E sets of hash values $\{h_j^i\}_{j\in[U]}$ with dummy verification points and random values, where $h_j^i := \mathsf{CH.Eval}(pk_j^i, dvp_j^i, rd_j^i)$ and $rd_j^i := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“DR”}||i)$;
- For $i \in [E]$, compute the i-th permutation key $k_p^i := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“PM”}||i)$ and permuted set $X_i = \{x_1^i, \ldots, x_U^i\} := \mathsf{PM.Shuffle}(k_p^i, \{1, \ldots, U\})$, where $x_j^i \in [U]$ for $j \in [U]$;
- Build E sub-Merkle trees $\text{SMT} = \{\text{MTI}_i\}_{i\in[E]}$, where $\text{MTI}_i := \mathsf{MT.Build}(V_i)$ and $V_i := \{h_{x_j^i}^i\}_{j\in[U]}$;
- Generate the group public key $\mathsf{gpk_G} = \mathsf{MT.Build}(\text{SMT})$.

Meanwhile, we define the group management message $\mathsf{GM_G}$ to store SMT, a list IDL_G recording the identities of registered group members, and a set of variables $\{\text{MPI}_{x_j^i} = (pk_{x_j^i}^i, C_{x_j^i}^i)\}_{j\in[U]}$, where each variable $\text{MPI}_{x_j^i}$ only stores elements of *current-active* verification epoch. We stress that the set $\{\text{MPI}_{x_j^i} = (pk_{x_j^i}^i, C_{x_j^i}^i)\}_{j\in[U]}$ is mutable. Therefore, RA should regularly remove those expired tuples in $\{\text{MPI}_{x_j^i} = (pk_{x_j^i}^i, C_{x_j^i}^i)\}_{j\in[U]}$, and also update it to include the tuples being used in the next valid verification epoch. We change the revocation list RL_G to store U 1-bit elements, each of which denotes whether or not the corresponding group member in IDL_G is revoked. The RL_G is initialized with zero bits.

Algorithm 1: $\mathsf{RASetup}(1^\kappa, \mathsf{setpms})$

Input: Parameters 1^κ and $\mathsf{setpms} = (U, T_s, T_e, \Delta_e, \Delta_s)$
Output: System parameters pms, group public key $\mathsf{gpk_G}$, initial group management message $\mathsf{GM_G}$, revocation list RL_G, and RA's secret key sk_{RA}

$T_s := T_{ct}$; // Set start time
$(\mathsf{pms}_{\mathsf{F}_1}, k_{\mathsf{RA}}) := \mathsf{F}_1.\mathsf{Setup}(1^\kappa)$; // Initialize PRF
$(\mathsf{pms}_{\mathsf{CRH}}, hk) \leftarrow \mathsf{H}_1.\mathsf{Setup}(1^\kappa)$; // Initialize collision-resistant hash function
$E := (T_e - T_s)/\Delta_e$; // Number of TOTP protocol instances
$\mathsf{pms} := (\mathsf{pms}_{\mathsf{F}_1}, \mathsf{pms}_{\mathsf{CRH}}, hk, \mathsf{setpms})$
for $j \in [U]$ **do**
  $ks_j := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||j)$; // Secret seed for member j
  for $i \in [E]$ **do**
    $dvp_j^i := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“DVP”}||i)$;
    $rk_j^i := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“CHR”}||i)$;
    $pk_j^i := \mathsf{CH.Setup}(1^\kappa; rk_j^i)$;
    $rd_j^i := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“DR”}||i)$;
    $h_j^i := \mathsf{CH.Eval}(pk_j^i, dvp_j^i, rd_j^i)$;
  end for
end for
for $i \in [E]$ **do**
  $k_p^i := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“PM”}||i)$;
  $X_i = \{x_1^i, \ldots, x_U^i\} := \mathsf{PM.Shuffle}(k_p^i, \{1, \ldots, U\})$; // Permuted set of indices, for $x_j^i \in [U]$ and $j \in [U]$
  $V_i := \{h_{x_j^i}^i\}_{j\in[U]}$; // Set of hash values
  $\text{MTI}_i := \mathsf{MT.Build}(V_i)$; // Build sub-Merkle tree
end for
$\mathsf{gpk_G} := \mathsf{MT.Build}(\{\text{MTI}_i\}_{i\in[E]})$; // Build group public key
$\mathsf{GM_G} := (\text{SMT}, \text{IDL}_\mathsf{G}, \{\text{MPI}_{x_j^i}\}_{j\in[U]}) = (\text{SMT}, \emptyset, \emptyset)$; // Group management message
$\text{RL}_\mathsf{G} := \{0\}^U$; // Initialize revocation list with zero bits
return pms, $\mathsf{gpk_G}$, $\mathsf{GM_G}$, RL_G, and sk_{RA}

– $\mathsf{PInit}(\mathsf{ID}_j)$: A group member ID_j executes $(\mathsf{pms}, kt_{\mathsf{ID}_j}) \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\kappa)$ to initialize a PRF, and the general parameters (i.e., T_i, Δ_e, and Δ_s) of TOTP. The secret key sk_{ID_j} is initially set as $sk_{\mathsf{ID}_j} := kt_{\mathsf{ID}_j}$, which will be updated after the join procedure.
– $\mathsf{Join}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, \mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$: We assume that all joined identities kept in IDL_G are sorted in their join time. Hence, RA can map a group member ID_j to a number $\alpha_{\mathsf{ID}_j} \in [U]$ according to its join order (hereafter α_{ID} will be referred to as a *transformed identity* of ID_j). Once the member's position in IDL_G is determined, it should not be changed during the whole life-span of the group. Namely, IDL_G is an append-only data structure. Let $\alpha_{\mathsf{ID}_j} = \text{IDL}_\mathsf{G}[\mathsf{ID}_j] \in [U]$ denote the position of ID_j in IDL_G. RA handles the enrollment of an unregistered party ID_j by generating a secret seed $ks_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||\alpha_{\mathsf{ID}_j})$ for ID_j, and appending ID_j to IDL_G and the tuple $(ks_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j})$ to $\mathsf{Ax}_{\mathsf{ID}_j}$, respectively. ID_j will include the tuple $(ks_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j})$ as a part of its secret

Algorithm 2: $\mathsf{PInit}(\mathsf{ID}_j)$

Input: Identity ID_j of a group member
Output: Secret key sk_{ID_j} and the initial verification state $\mathsf{vst}_{\mathsf{ID}_j}$

$(\mathsf{pms}, kt_{\mathsf{ID}_j}) := \mathsf{F}_2.\mathsf{Setup}(1^\kappa)$;
$sk_{\mathsf{ID}_j} := kt_{\mathsf{ID}_j}$;
$\mathsf{vst}_{\mathsf{ID}_j} := \emptyset$;
return sk_{ID_j} and $\mathsf{vst}_{\mathsf{ID}_j}$;

key sk_{ID_j}. Note that $\mathsf{vst}_{\mathsf{ID}_j}$ can be empty in this algorithm. Namely, we allow a post-authorization strategy to bind a verification point of a party during the usage of passwords.

Algorithm 3: $\mathsf{Join}(sk_{\mathsf{RA}}, \mathsf{gpk}_\mathsf{G}, \mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$

Input: RA's secret key sk_{RA}, group public key gpk_G, joining party's identity ID_j and its initial verification state $\mathsf{vst}_{\mathsf{ID}_j}$
Output: Auxiliary information $\mathsf{Ax}_{\mathsf{ID}_j}$ to ID_j, and updated group management message GM'_G

Map ID_j to $\alpha_{\mathsf{ID}_j} \in [U]$ based on its join order ;
Generate secret seed: $ks_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||\alpha_{\mathsf{ID}_j})$;
Append ID_j to IDL_G ; `// Update` IDL_G `in` GM_G
$\mathrm{IDL}_\mathsf{G}[\mathsf{ID}_j] := \alpha_{\mathsf{ID}_j}$;
Append $(ks_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j})$ to $\mathsf{Ax}_{\mathsf{ID}_j}$; `//` ID_j `updates` sk_{ID_j} `by appending` $(ks_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j})$
return $\mathsf{Ax}_{\mathsf{ID}_j}$ and GM'_G

– $\mathsf{GetSD}(sk_{\mathsf{ID}_j}, T)$: If the i-th TOTP instance is not initialized, then ID_j first runs $\mathsf{pms}^i_{\mathsf{TOTP}} := \mathsf{TOTP.Setup}(1^\kappa, T_i, T_i + \Delta_e, \Delta_s)$, where $T_i = T_{i-1} + \Delta_e$ and $T_0 := T_s$. This algorithm computes i-th password seed as $sd^i_{\mathsf{ID}_j} := \mathsf{F}_2(kt_{\mathsf{ID}_j}, \mathsf{ID}_j||i)$.

Algorithm 4: $\mathsf{GetSD}(sk_{\mathsf{ID}_j}, T)$

Input: ID_j's secret keysk_{ID_j} and time T
Output: Secret seed $sd^i_{\mathsf{ID}_j}$

if *i-th TOTP instance is not initialized* **then**
  $T_i := T_{i-1} + \Delta_e$, $T_0 := T_s$;
  $\mathsf{pms}^i_{\mathsf{TOTP}} := \mathsf{TOTP.Setup}(1^\kappa, T_i, T_i + \Delta_e, \Delta_s)$;
end if
$sd^i_{\mathsf{ID}_j} := \mathsf{F}_2(kt_{\mathsf{ID}_j}, \mathsf{ID}_j||i)$;
return $sd^i_{\mathsf{ID}_j}$

– $\mathsf{PwGen}(sd^i_{\mathsf{ID}_j}, T_{ct})$: ID_j carries out the following steps to generate a password $pw^{i,z}_{\mathsf{ID}_j} = (\bar{pw}^{i,z}_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j}, C^i_{\alpha_{\mathsf{ID}_j}})$:
  - Compute the corresponding TOTP password $\bar{pw}^{i,z}_{\mathsf{ID}_j} := \mathsf{TOTP.PGen}(sd^i_{\mathsf{ID}_j}, T_{ct})$, where $z := \frac{\lceil T_{ct} - T_s - i \cdot \Delta_e \rceil}{\Delta_s}$ is password index in the i-th verification epoch;
  - Derive the i-th ASE key $ke^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“KG”}||i)$, and a random value $re^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“ER”}||i)$ to encrypt ID_j resulting in the identity ciphertext $C^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{ASE.Enc}(ke^i_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j}; re^i_{\alpha_{\mathsf{ID}_j}})$;
  - Compute the i-th random key $rk^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \mathsf{G}||\text{“CHR”}||i)$ and get the chameleon secret key $sk^i_{\alpha_{\mathsf{ID}_j}}$ from $rk^i_{\alpha_{\mathsf{ID}_j}}$;
  - Compute a new variant of the verification point $\hat{vp}^i_{\mathsf{ID}_j} := \mathsf{H}_1(vp^i_{\mathsf{ID}_j}||C^i_{\alpha_{\mathsf{ID}_j}}||i)$ and the corresponding collision $r^i_{\mathsf{ID}_j} := \mathsf{CH.Coll}(sk^i_{\alpha_{\mathsf{ID}_j}}, dvp^i_{\alpha_{\mathsf{ID}_j}}, rd^i_{\alpha_{\mathsf{ID}_j}}, \hat{vp}^i_{\mathsf{ID}_j})$, where $dvp^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \mathsf{G}||\text{“DVP”}||i)$ and $rd^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \mathsf{G}||\text{“DR”}||i)$.
– $\mathsf{Verify}(\mathsf{gpk}_\mathsf{G}, pw^{i,z}_{\mathsf{ID}_j}, T, \mathrm{RL}_\mathsf{G})$: RA should first handle *outsourced proof generation* procedure regularly (i.e., via Algorithm 6) for the group members and the verifier. Specifically, at the end of (i-1)-th verification epoch, RA generates and publishes i-th permuted sets $V_i = \{h^i_{x^i_j}\}$ and $\{\mathrm{MPI}_{x^i_j}\}_{j \in [U]}$, and the Merkle proof $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$. RA is able to compute $\mathrm{MPI}_{x^i_j}$ based on her secret seed k_{RA}, the permuted set X_i, and the registration identity

Algorithm 5: $\mathsf{PwGen}(sd^i_{\mathsf{ID}_j}, T_{ct})$

Input: ID_j's i-th secret seed $sd^i_{\mathsf{ID}_j}$ and the current time T_{ct}
Output: Password $pw^{i,z}_{\mathsf{ID}_j}$

$i := \lfloor \frac{T_{ct}-T_s}{\Delta_e} \rfloor$; // Epoch index
$z := \frac{\lceil T_{ct}-T_s-i\cdot\Delta_e \rceil}{\Delta_s}$; // Password index in the i-th TOTP chain
$p\bar{w}^{i,z}_{\mathsf{ID}_j} := \mathsf{TOTP.PGen}(sd^i_{\mathsf{ID}_j}, T_{ct})$; // Password in the i-th TOTP instance
$ke^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“KG”}||i)$;
$re^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“ER”}||i)$;
$C^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{ASE.Enc}(ke^i_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j}; re^i_{\alpha_{\mathsf{ID}_j}})$; // Identity ciphertext
$dvp^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \mathsf{G}||\text{“DVP”}||i)$; // The dummy verification point corresponds to α_{ID_j} in the i-th epoch
$rd^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \mathsf{G}||\text{“DR”}||i)$; // The dummy randomness corresponds to α_{ID_j} in the i-th epoch
$rk^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \mathsf{G}||\text{“CHR”}||i)$;
$sk^i_{\alpha_{\mathsf{ID}_j}} \leftarrow rk^i_{\alpha_{\mathsf{ID}_j}}$; // E.g., $sk^i_{\alpha_{\mathsf{ID}_j}} = rk^i_{\alpha_{\mathsf{ID}_j}}$ for the discrete-log based chameleon hash function [40]
$\hat{vp}^i_{\mathsf{ID}_j} := \mathsf{H}_1(vp^i_{\mathsf{ID}_j}||C^i_{\alpha_{\mathsf{ID}_j}}||i)$;
$r^i_{\mathsf{ID}_j} := \mathsf{CH.Coll}(sk^i_{\alpha_{\mathsf{ID}_j}}, dvp^i_{\alpha_{\mathsf{ID}_j}}, rd^i_{\alpha_{\mathsf{ID}_j}}, \hat{vp}^i_{\mathsf{ID}_j})$;
$pw^{i,z}_{\mathsf{ID}_j} = (p\bar{w}^{i,z}_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j}, C^i_{\alpha_{\mathsf{ID}_j}})$;
return $pw^{i,z}_{\mathsf{ID}_j}$

list $\mathrm{IDL}_{\mathsf{G}}$. The verifier can download supplementary verification materials V_i, $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$, and $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$ when i-th verification epoch starts. On receiving a password $(p\bar{w}^{i,z}_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j}, C^i_{\alpha_{\mathsf{ID}_j}})$, the verifier does the following steps:

- Return 0, if $C^i_{\alpha_{\mathsf{ID}_j}}$ is not contained in $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$;
- Get the position x such that MPI_x contains the identity ciphertext $C^i_{\alpha_{\mathsf{ID}_j}}$;
- Compute Merkle proof $\mathrm{Pf}^{\mathrm{MTI}_i}_{vp^i_{\mathsf{ID}_j}}$ based on $h^i_x \in V_i$ and V_i, the verification point $vp^i_{\mathsf{ID}_j}$ from $p\bar{w}^{i,z}_{\mathsf{ID}_j}$ and T, and $\hat{vp}^i_{\mathsf{ID}_j} := \mathsf{H}_1(vp^i_{\mathsf{ID}_j}||C^i_{\alpha_{\mathsf{ID}_j}}||i)$;
- Assemble the Merkle proof $\mathrm{Pf}^i_{\mathsf{ID}_j} := (\mathrm{Pf}^{\mathrm{MTI}_i}_{vp^i_{\mathsf{ID}_j}}, \mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i})$ for verifying the password under $\mathsf{gpk}_{\mathsf{G}}$;
- Return 0, if $h^i_y \neq \mathsf{CH.Eval}(pk^i_x, \hat{vp}^i_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j})$;
- Output 1, if and only if $\mathsf{MT.Verify}(\mathsf{gpk}_{\mathsf{G}}, h^i_x, \mathrm{Pf}^i_{\mathsf{ID}_j}) = 1$ and $\mathsf{TOTP.Verify}(vp^i_{\mathsf{ID}_j}, p\bar{w}^{i,z}_{\mathsf{ID}_j}, T) = 1$.

– $\mathsf{Revoke}(sk_{\mathsf{RA}}, \mathsf{ID}_j, \mathsf{gpk}_{\mathsf{G}}, \mathrm{RL}_{\mathsf{G}}, \mathrm{GM}_{\mathsf{G}})$: Let $\mathrm{IDL}_{\mathsf{G}}[\mathsf{ID}_j]$ denote the bit that corresponds to ID_j. To revoke a group member ID_j, RA sets $\mathrm{RL}_{\mathsf{G}}[\mathsf{ID}_j] := 1$, and stops updating MPI relevant to ID_j in all future verification epochs.
– $\mathsf{Open}(sk_{\mathsf{RA}}, \mathsf{gpk}_{\mathsf{G}}, pw^{i,z\prime}_{\mathsf{ID}_j}, T)$: This algorithm requires a updated password $pw^{i,z\prime}_{\mathsf{ID}_j} = (pw^{i,z}_{\mathsf{ID}_j}, pk^i_{\alpha_{\mathsf{ID}_j}}, \mathrm{Pf}^i_{\mathsf{ID}_j})$. RA does the following steps to reveal the identity of a group member:
  - Derive the verification point $vp^i_{\mathsf{ID}_j}$ from $p\bar{w}^{i,z}_{\mathsf{ID}_j}$ and T, and compute the variant of verification point $\hat{vp}^i_{\mathsf{ID}_j} := \mathsf{H}_1(vp^i_{\mathsf{ID}_j}||C^i_{\alpha_{\mathsf{ID}_j}}||i)$ and $h^i_y := \mathsf{CH.Eval}(pk^i_{\alpha_{\mathsf{ID}_j}}, \hat{vp}^i_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j})$;
  - Abort, if either $\mathsf{TOTP.Verify}(vp^i_{\mathsf{ID}_j}, p\bar{w}^{i,z}_{\mathsf{ID}_j}, T) = 0$ or $\mathsf{MT.Verify}(\mathsf{gpk}_{\mathsf{G}}, h^i_y, \mathrm{Pf}^i_{\mathsf{ID}_j}) = 0$;
  - Compute the permutation key $k^i_p := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“PM”}||i)$ and the permuted set $X_i = \{x^i_1, \ldots, x^i_U\} := \mathsf{PM.Shuffle}(k^i_p, \{1, \ldots, U\})$, where $x^i_j \in [U]$ for $j \in [U]$;
  - Get the value of y in terms of the position of h^i_y in the Merkle proof $\mathrm{Pf}^i_{\mathsf{ID}_j}$, and then obtain the transformed identity $\alpha_{\mathsf{ID}_j} = x^i_y$;
  - Retrieve $\mathrm{IDL}_{\mathsf{G}}$ using α_{ID_j} to find the identity ID_j.

  Nevertheless, RA still needs to confirm it. RA computes the secret seed $ks_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||\alpha_{\mathsf{ID}_j})$ and the encryption key $ke^i_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“KG”}||i)$ to decrypt the transformed identity $\alpha'_{\mathsf{ID}_j} := \mathsf{ASE.Dec}(ke, C^i_{\alpha_{\mathsf{ID}_j}})$. If $\alpha'_{\mathsf{ID}_j} = \alpha_{\mathsf{ID}_j}$, then RA returns ID_j.

Main Steps for Executing DGTOne. We also high-levelly show the seven main executing steps of DGTOne in Figure 9. Initially, RA generates the parameters in setpms in terms of a specific application (as exemplified in Section 5) and runs the initialization algorithm $\mathsf{DGTOne.RASetup}(1^\kappa, \mathsf{setpms})$, as depicted in Figure 9.(1), to particular creates a group public key $\mathsf{gpk}_{\mathsf{G}}$ for a group G. $\mathsf{gpk}_{\mathsf{G}}$ is mainly generated as the root of a Merkle tree, based on a set of dummy verification points $\{dvp^i_j\}_{j\in[U]}$ organized by verification epochs. After this, each group member ID_j can initialize (Figure 9.(2)) its secret key sk_{ID_j} and local parameters by running $\mathsf{DGTOne.PInit}(\mathsf{ID}_j)$ to prepare for registration. Then, ID_j can send a registration request to RA which will run (Figure 9.(3)) the

Algorithm 6: $\mathsf{RAProofGen}(sk_{\mathsf{RA}}, i, \mathsf{GM_G})$

Input: RA's secret key sk_{RA}, sub-Merkle tree SMT, and group management message $\mathsf{GM_G}$
Output: Leaves $V_i = \{h^i_{x^i_j}\}$ and corresponding $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$, and Merkle proof $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$

for $j \in [U]$ **do**
  $ks_j := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||j)$; `// Secret seed for member j`
  $dvp^i_j := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“DVP”}||i)$;
  $rd^i_j := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“DR”}||i)$;
  $rk^i_j := \mathsf{F}_1(ks_j, \mathsf{G}||\text{“CHR”}||i)$;
  $pk^i_j := \mathsf{CH.Setup}(1^\kappa; rk^i_j)$;
  $h^i_j := \mathsf{CH.Eval}(pk^i_j, dvp^i_j, rd^i_j)$;
  $ke^i_j := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“KG”}||i)$;
  $re^i_j := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“ER”}||i)$;
  if *the j is smaller than or equal to the number of registered identities in IDL_{G}* **then**
    $C^i_j := \mathsf{ASE.Enc}(ke^i_j, j; re^i_{\alpha_{\mathsf{ID}_j}})$
  else
    $C^i_j := \mathsf{ASE.Enc}(ke^i_{\alpha_{\mathsf{ID}_j}}, 0; re^i_{\alpha_{\mathsf{ID}_j}})$
  end if
end for
$k^i_p := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“PM”}||i)$;
$X_i = \{x^i_1, \ldots, x^i_U\} := \mathsf{PM.Shuffle}(k^i_p, \{1, \ldots, U\})$;
$V_i := \{h^i_{x^i_j}\}_{j\in[U]}$;
$\{\mathrm{MPI}_{x^i_j} = (pk^i_{x^i_j}, C^i_{x^i_j})\}_{j\in[U]}$;
Compute $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$ from SMT ;
return V_i, $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$, and $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$

Algorithm 7: $\mathsf{Verify}(\mathsf{gpk_G}, pw^{i,z}_{\mathsf{ID}_j}, T, \mathrm{RL_G})$

Input: Group public key $\mathsf{gpk_G}$, password $pw^{i,z}_{\mathsf{ID}_j} = (p\bar{w}^{i,z}_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j}, C^i_{\alpha_{\mathsf{ID}_j}})$, verification time T, and revocation list $\mathrm{RL_G}$
Output: 0 or 1

Download V_i, $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$, and $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$ from RA if necessary ; `// If they have not been obtained yet for the current epoch`
if $C^i_{\alpha_{\mathsf{ID}_j}} \notin \{MPI_{x^i_j}\}_{j\in[U]}$ **then**
  return 0 ;
Get position x such that MPI_x contains $C^i_{\alpha_{\mathsf{ID}_j}}$;
$h^i_x \leftarrow V_i$;
Compute the verification point $vp^i_{\mathsf{ID}_j}$ from $p\bar{w}^{i,z}_{\mathsf{ID}_j}$ and T ; `// Following the specification of TOTP`
$\hat{vp}^i_{\mathsf{ID}_j} := \mathsf{H}_1(vp^i_{\mathsf{ID}_j}||C^i_{\alpha_{\mathsf{ID}_j}}||i)$;
if $h^i_x \neq \mathsf{CH.Eval}(pk^i_x, \hat{vp}^i_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j})$ **then**
  return 0 ;
Compute Merkle proof $\mathrm{Pf}^{\mathrm{MTI}_i}_{vp^i_{\mathsf{ID}_j}}$ based on h^i_x and V_i;
Assemble $\mathrm{Pf}^i_{\mathsf{ID}_j} := (\mathrm{Pf}^{\mathrm{MTI}_i}_{vp^i_{\mathsf{ID}_j}}, \mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i})$;
if $\mathsf{MT.Verify}(\mathsf{gpk_G}, h^i_x, Pf^i_{\mathsf{ID}_j}) = 1$ ***and*** $\mathsf{TOTP.Verify}(vp^i_{\mathsf{ID}_j}, p\bar{w}^{i,z}_{\mathsf{ID}_j}, T) = 1$ **then**
  return 1 ;
return 0 ;

Algorithm 8: $\mathsf{Revoke}(sk_{\mathsf{RA}}, \mathsf{ID}_j, \mathsf{gpk_G}, \mathrm{RL_G}, \mathsf{GM_G})$

Input: sk_{RA}, ID_j, $\mathsf{gpk_G}$, $\mathrm{RL_G}$, $\mathsf{GM_G}$
Output: Updated $\mathrm{RL_G}$

Procedure: Revoking a group member ID_j
begin
  Locate the position of ID_j in $\mathrm{IDL_G}$;
  Set $\mathrm{RL_G}[\mathsf{ID}_j] := 1$ to mark ID_j as revoked ;
  Stop updating the registration status MPI related to ID_j for all future verification epochs ;
  return Updated $\mathrm{RL_G}$;
end

Algorithm 9: $\mathsf{Open}(sk_{\mathsf{RA}}, \mathsf{gpk}_{\mathsf{G}}, pw^{i,z\,\prime}_{\mathsf{ID}_j}, T)$

Input: sk_{RA}, $\mathsf{gpk}_{\mathsf{G}}$, updated password $pw^{i,z\,\prime}_{\mathsf{ID}_j} = (pw^{i,z}_{\mathsf{ID}_j}, pk^{i}_{\alpha_{\mathsf{ID}_j}}, \mathrm{Pf}^{i}_{\mathsf{ID}_j})$, T
Output: Identity ID_j of group member

Derive the verification point $vp^{i}_{\mathsf{ID}_j}$ from $\bar{pw}^{i,z}_{\mathsf{ID}_j}$ and T ;
Compute the variant verification point $\hat{vp}^{i}_{\mathsf{ID}_j} := \mathsf{H}_1(vp^{i}_{\mathsf{ID}_j} || C^{i}_{\alpha_{\mathsf{ID}_j}} || i)$;
Compute $h^{i}_{y} := \mathsf{CH.Eval}(pk^{i}_{\alpha_{\mathsf{ID}_j}}, \hat{vp}^{i}_{\mathsf{ID}_j}, r^{i}_{\mathsf{ID}_j})$;
Abort if $\mathsf{TOTP.Verify}(vp^{i}_{\mathsf{ID}_j}, \bar{pw}^{i,z}_{\mathsf{ID}_j}, T) = 0$ or $\mathsf{MT.Verify}(\mathsf{gpk}_{\mathsf{G}}, h^{i}_{y}, \mathrm{Pf}^{i}_{\mathsf{ID}_j}) = 0$;
Compute the permutation key $k^{i}_{p} := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“PM”}||i)$;
Generate the permuted set $X_i = \{x^{i}_{1}, \ldots, x^{i}_{U}\} := \mathsf{PM.Shuffle}(k^{i}_{p}, \{1, \ldots, U\})$;
Identify y from the position of h^{i}_{y} in the Merkle proof $\mathrm{Pf}^{i}_{\mathsf{ID}_j}$;
Set the transformed identity $\alpha_{\mathsf{ID}_j} = x^{i}_{y}$;
Retrieve $\mathrm{IDL}_{\mathsf{G}}$ using α_{ID_j} to find the identity ID_j ;
Compute the secret seed $ks_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(k_{\mathsf{RA}}, \mathsf{G}||\text{“KS”}||\alpha_{\mathsf{ID}_j})$;
Compute the encryption key $ke^{i}_{\alpha_{\mathsf{ID}_j}} := \mathsf{F}_1(ks_{\alpha_{\mathsf{ID}_j}}, \text{“KG”}||i)$;
Decrypt the transformed identity $\alpha'_{\mathsf{ID}_j} := \mathsf{ASE.Dec}(ke^{i}_{\alpha_{\mathsf{ID}_j}}, C^{i}_{\alpha_{\mathsf{ID}_j}})$;
Return ID_j if $\alpha'_{\mathsf{ID}_j} = \alpha_{\mathsf{ID}_j}$;

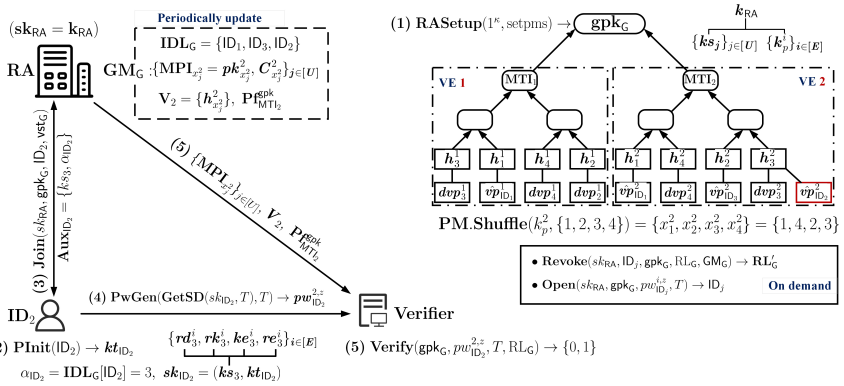

Fig. 9: Overview of DGTOne with $U = 4$ and $E = 2$. Assuming that ID_3 and ID_2 join the group in the 2-th verification epoch (VE).

$\mathsf{DGTOne.Join}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, \mathsf{ID}_j, \mathsf{vst}_{\mathsf{ID}_j})$ to handle the enrollment of the group member ID_j. RA would update $\mathsf{GM_G}$ accordingly (to mainly append its transformed identity α_{ID_j} into the registration list $\mathrm{IDL_G}$) and return the generated secrets $(ks_{\alpha_{\mathsf{ID}_j}}, \alpha_{\mathsf{ID}_j})$ to ID_j. The registered party ID_j can then use (Figure 9.(4)) the passwords generated by $\mathsf{DGTOne.PwGen}(sd^i_{\mathsf{ID}_j}, T_{ct})$ at any verifiers (determined by specific applications), where the i-th password seed $sd^i_{\mathsf{ID}_j}$ is computed by running the algorithm $\mathsf{GetSD}(sk_{\mathsf{ID}_j}, T_{ct})$. To verify a password $pw^{i,z}_{\mathsf{ID}_j} = (p\bar{w}^{i,z}_{\mathsf{ID}_j}, r^i_{\mathsf{ID}_j}, C^i_{\alpha_{\mathsf{ID}_j}})$, the verifier should first download (Figure 9.(5)) the supplementary verification materials V_i, $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$ from RA at the beginning of the i-th verification epoch related to $pw^{i,z}_{\mathsf{ID}_j}$. Then, the verifier can run the verification algorithm $\mathsf{DGTOne.Verify}(\mathsf{gpk_G}, pw^{i,z}_{\mathsf{ID}_j}, T, \mathrm{RL_G})$ to check the password (also shown in Figure 9.(5)).

Additionally, there are two steps executed upon special demands. A party may ask RA to withdraw its membership of the group G. Upon receiving the revocation request, RA runs (Figure 9.(6)) the revocation algorithm $\mathsf{DGTOne.Revoke}(sk_{\mathsf{RA}}, \mathsf{ID}_j, \mathsf{gpk_G}, \mathrm{RL_G}, \mathsf{GM_G})$ to revoke a party ID_j. This step is almost computationally free since RA only needs to change a bit of the revocation list to $\mathrm{RL_G}[\mathsf{ID}_j] := 1$, and stop updating MPI for ID_j thereafter. When necessary, at the request of a third party (e.g., hospital and judicial institution), RA can open (Figure 9.(7)) the identity of the generator of a password by running $\mathsf{DGTOne.Open}(sk_{\mathsf{RA}}, \mathsf{gpk_G}, pw^{i,z\prime}_{\mathsf{ID}_j}, T)$.

Remarks. For usage privacy, we require the verifier to download the Merkle proof generation ingredients of the current-active verification epoch, i.e., V_i, $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$, and $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$, in the algorithm Verify. Although RA can generate the complete Merkle proof $\mathrm{Pf}^i_{\mathsf{ID}_j}$ for the verifier once she receives the identity-ciphertext $C^i_{\alpha_{\mathsf{ID}_j}}$ as a query token, she would know the usage status of ID_j's password. Because each group member may not want RA to know with whom she is communicating. Fortunately, the verifier can download those ingredients once and use them to verify all passwords in that verification epoch. If the storage is allowed, RA can also compute those ingredients of multiple upcoming verification epochs to ease the download frequency. Besides, each group does not have any interaction with RA during the password usage phase, no privacy on the password usage status is leaked to RA.

Correctness. Generally speaking, the correctness of DGTOne is guaranteed by that of its building blocks. First, an honest group member's password $p\bar{w}^{i,z}_{\mathsf{ID}_j}$ is generated from a TOTP instance, so its verification correctness is ensured by that of the underlying TOTP scheme. Furthermore, the verification point of such a TOTP password $p\bar{w}^{i,z}_{\mathsf{ID}_j}$ is uniquely bound to a leaf of the Merkle tree $\mathsf{gpk_G}$ via the collision feature of the chameleon hash function CH. The security and the correctness of CH and the Merkle tree can guarantee an honest verification point will be correctly verified. Second, as the registration authority is honest, all identity ciphertexts $\{C^i_{\alpha_{\mathsf{ID}_j}}\}_{\mathsf{ID}_j\in\mathrm{HML}}$ are encrypted by RA honest without ambiguity. Therefore, the correctness of ASE ensures that each identity ciphertext of an honest group member can be decrypted to its identity.

4.3 Security Analysis

Theorem 1. *Assuming that the building blocks* TOTP, H_1, $\{\mathsf{F}_i\}_{i\in[2]}$, MT, ASE, *and* PM *are secure, then* DGTOne *provides anonymity.*

Here, we briefly introduce the high-level idea behind the proof of Theorem 1. The full proof is presented in the supplementary document (Appendix C). Basically, the security of PRF, ASE, and H_1 ensures that a PPT adversary cannot infer any useful information regarding the challenge TOTP seeds from the outputs of these building blocks with non-negligible probabilities. We can first change the outputs of pseudo-random function families $\{\mathsf{F}_i\}$ to truly random values to exclude the collision among secret keys of group members and enable the security reduction to other building blocks (e.g., ASE and CH). The security of CH implies that each verification point is uniquely bound to a leaf of Merkle tree gpk. Without the encryption key of ASE, the adversary cannot decrypt the identity ciphertext. Moreover, the mapping relation between a verification point and a leaf is random, so the adversary cannot gain a non-negligible advantage over 1/2 to guess the identity linked to a leaf.

Theorem 2. DGTOne *provides traceability under the same assumptions of Theorem 1.*

The security proof of this theorem is similar to that of Theorem 1. Nevertheless, we just additionally reduce the unforgeability of DGTOne to that of the TOTP scheme and the Merkle tree, respectively. The proof of Theorem 2 is included in the supplementary document (Appendix D).

5 Use Cases

In the following, we list a few (but not limited to) typical use cases of DGTOTP. We are going to show that, unlike other password-based anonymous authentication schemes (e.g., [58,24,57,49]), DGTOTP can be not only

used as a building block to construct secure application protocols (in the first two use cases) but also used independently (in the third one). In the first use case, we will also discuss the limitations and solutions for enhancing the security of DGTOTP in practice.

Anonymous-Client Authenticated Confidential Channel Establishment. Unlike the typical application of TOTP (e.g., [42,39]) that authenticates a party (as the second authentication factor) to a fixed verifier (at which the party is registered), DGTOTP is designed to be a public verifiable authentication factor. Namely, a password of a DGTOTP scheme can be verified by an arbitrary verifier at a time. We will describe our idea in conjunction with the traditional password-based client authentication over Transport Layer Security Protocols (TLS) [25,58], where TLS can only provide server authentication only. Here, we formally consider TLS as a server-only authenticated confidential channel establishment (S-ACCE) protocol [18,43,21]. That is, we will design a client-anonymous authentication scheme based on S-ACCE to yield an anonymous-client ACCE (ACACCE) scheme. Note that the verifier just takes the role of 'server' in S-ACCE, but it does not have to be a real web server – namely, the verifier could be any entities (e.g., roadside unit in V2X) due to the public verifiability of DGTOTP. Once such a secure channel is established, the group member can continuously send any messages (e.g., printing or psychological counseling data) over that channel efficiently and anonymously.

However, naively sending a (D)GTOTP password over TLS may be subject to password replay attacks. That is, a malicious verifier receiving a DGTOTP password can use it at other verifiers. To avoid such a security issue, the first application of GTOTP in [56] adopts a commitment-based secure application scenario in which a group member commits a message using a GTOTP password as a commitment key and opens the password after it was expired, so the password cannot be abused anymore. However, an obvious shortcoming of such an application is the authentication latency. Here, we are motivated to provide a solution for securely using DGTOTP passwords in real-time. Our idea is to use introduce a third-party and semi-honest authentication server (AS) to ensure a one-time usage of a DGTOTP password. Meanwhile, we need a hash function $\mathsf{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ to implement a commitment scheme. We shall use the AS's authenticated public database (e.g., each item is digitally signed by the AS) to record the usage status (i.e., a hash of the password and a one-time pseudonym of the verifier) of a GTOTP password. However, we only require the AS to provide a 'time-based append-only' feature for efficiency. That is, the AS only needs to guarantee that an added item will be stored without being modified or deleted for a short time (e.g., the life span Δ_s of a password). Before verifying a password, an honest verifier should check the corresponding usage status on the AS. I.e., if its usage status is stored on the AS, the verifier would reject the session immediately; otherwise, it should generate and upload the usage status to the AS.

The above resulting scheme can be used as a privacy-preserving federated identity management system (note that regular FIMS also involves an AS). We present the detailed steps of the above security enhancement solution in the supplementary document (Appendix E).

Privacy-Preserving Proof of Location. In [56], Yang et al. have demonstrated that the GTOTP can be used as a building block to construct distributed privacy-preserving proof of location (PPPoL) schemes. A PPPoL scheme itself finds application in numerous real-world scenarios, such as secure measurement of the location proximity of a (say borrowed) mobile device to the local network [53] and location attestation in Vehicular Ad Hoc Networks [7]. A simple way to improve Yang et al.'s PPPoL scheme is to replace YJN+ with DGTOne to support dynamic group management. However, another drawback of Yang et al.'s scheme is the long delay in the location proof verification. As aforementioned, Yang et al.'s scheme leverages a GTOTP-based commitment scheme to authenticate location proofs of witnesses. Consequently, the committed location proofs can only be verified after the corresponding passwords are expired, introducing a long delay in verification. There's potential to alleviate this delay by applying our above DGTOTP based ACACCE scheme to boost the message authentication and verification involved in the Yang et al. scheme. We leave such a straightforward replacement as future work.

Privacy-preserving Contact Tracing. We notice that DGTOTP can also be used alone to realize approximately private automated contact tracing for pandemics (e.g., Covid-19), as long as the majority of the participants are honest.Specifically, users can keep sending and collecting DGTOTP passwords with short-range communication methods (e.g., Bluetooth). Once a patient is confirmed, RA can publish its historical DGTOTP passwords (together with many dummy passwords for privacy) so that others can check whether they received his/her passwords or not. Thanks to the traceability of DGTOTP, the identities of contacts can also be revealed by RA from the passwords (unlike other decentralized private contact tracing scheme [47,44]) to alert them of a health check. Meanwhile, the privacy of all contacts (incl. the patient) can be preserved to each other because of the anonymity of passwords.

6 Evaluation

In this section, we experimentally evaluate the performance of our proposed DGTOTP scheme DGTOne. We also include a comparison to the static GTOTP scheme YJN+ [56].

Experiment Setup. We implemented DGTOne with the Bouncy Castle FIPS Java API [20]. The source codes of our implementation are made available at [1]. Our experiments are conducted respectively on a 64-bit machine with an Intel(R) i7-10700 at 2.90GHz at 2.2 GHz and 16 GB RAM to benchmark the algorithms (i.e., RASetup, Join, Revoke, Open, DGTOne.Verify) of RA and verifier, and on mobile phone with Snapdragon 855 CPU (up to 2.84GHz) and 8GB RAM to benchmark the algorithms (i.e., PInit, GetSD and PwGen) of a group member. In our implementation, we instantiate the TOTP scheme with the T/Key [39]. The hash functions H_1 and H_2 are implemented with SHA256. We implemented the chameleon hash function using the discrete logarithm based one in [40] (but using the NIST elliptic curve P-256 for the implementation), which has a fast collision generation algorithm. The prime numbers (p, q) used in the chameleon hash function are set to be 256-bit and 3072-bit (respectively) for providing 128-bit security [3]. We also use AES-GCM-SIV [28] with 128-bit key to implement the authenticated symmetric encryption ASE. The permutation scheme PM is implemented by the Shuffle function of Java.

For comparison, we adopt similar TOTP parameters as in [56] in the experiments. Specifically, we set the life span $\Delta_e = 300$ seconds (s) for each verification point and the password generation interval $\Delta_s = 5$s, which lead to the passwords in a chain $N = \frac{\Delta_e}{\Delta_s} = 60$. The parameters U and E are variables. We just benchmark DGTOne with a proof-of-concept prototype without considering any optimizations utilizing all available CPU cores.

6.1 Performance of DGTOne

RA Setup Time. The setup time of RA is determined by the parameters U and E, respectively. We show the concrete costs of DGTOne.RASetup in Figure 10 (a). Since the DGTOne.RASetup can be run at any time by RA (which is also powerful), it is still practical.

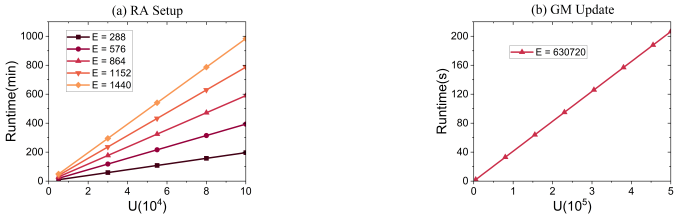

Fig. 10: Runtimes of RA Setup and GM Update.

Group Member Initialization Time. The DGTOne.PInit runs only one Setup algorithm of PRF. Since PRF is implemented by AES with 128-bit key, the setup of AES needs 23.4 microseconds (μs) on the mobile phone.

TOTP Seed Generation Time. The TOTP secret seed generation runs one PRF evaluation. To meet the secret seed of TOTP, a group member needs to run twice the encryption algorithm of AES to generate a 256-bit secret seed of the TOTP scheme. Hence, the runtime of GetSD is 1.8 μs.

Join Time. The cost of the join procedure is constant as well, which is dominated by one encryption algorithm of AES to generate a 128-bit secret key of a group member. As a result, the runtime of DGTOne.Join is approximately $0.9\mu s$.

Password Generation Time. For simplicity, we benchmark the runtime of DGTOne.PwGen without caching any TOTP passwords. However, each group member can cache the TOTP secret seed, identity ciphertext, and the chameleon hash collision to save computational costs for continuous password usage. In the worst case, a group member needs to run $N = 60$ times of H_1 to go through the whole chain to get the first password being used in one verification epoch. Note that the computation of each element in the set $\{ke^i_{\mathsf{ID}_j}, re^i_{\mathsf{ID}_j}, dvp^i_{\alpha_{\mathsf{ID}_j}}, rd^i_{\alpha_{\mathsf{ID}_j}}\}$ needs two AES encryptions to meet the corresponding spaces. Nevertheless, a group member only needs to run $N/2$ hash evaluations in the average case with caching materials. Overall, the worst-case costs of DGTOne.PwGen requires 10 AES encryptions, 1 AES-GCM-SIV encryption, 62 hashes, and 1 chameleon hash collision evaluation. The runtimes of the password generation algorithm DGTOne.PwGen in the worst case and the average case are $152\mu s$ and 105.5 μs (respectively), which are practical. Of course, each group member can pre-generate the passwords being used in the next verification epoch during her idle time as in [39,56] to boost password generation. We do not take into account such a time-space trade-off since the storage cost is considered a priority.

Group Management Message Update Time. For verification, RA is required to update V_i, $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$, and $\mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}$ in the group management message GM_G. However, we assume that RA will cache the most expensive computations (i.e., the generation of sub-Merkle trees) to save time and compute the less costly ones on the fly. So RA can cache all roots of sub-Merkle trees and the corresponding Merkle proofs $\{\mathrm{MTI}_i, \mathrm{Pf}^{\mathsf{gpk}}_{\mathrm{MTI}_i}\}_{i\in[E]}$. The computations of V_i and $\{\mathrm{MPI}_{x^i_j}\}_{j\in[U]}$ involve 1 permutation, in the worst case, U chameleon hash setups and

evaluations, $9U + 1$ AES encryptions, and U encryptions of ASE. The runtime of group management message update time is shown in Figure 10 (b). For simplicity, we assume that the RA only cache the roots of sub-Merkle trees.

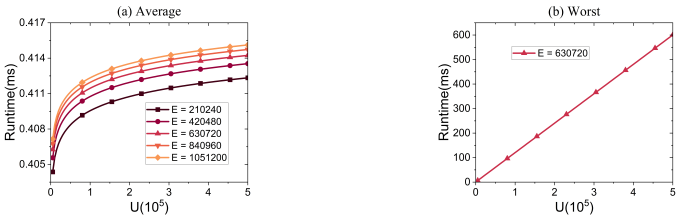

Fig. 11: Runtimes of Verification.

Password Verification Time. Similar to the password generation, the performance of DGTOne.Verify needs to compute $N = 60$ hash evaluations in the worst case, and compute $N/2$ times on average in the TOTP verification. Meanwhile, it needs 1 hash evaluation and 1 chameleon hash evaluation to compute the Merkle tree leaf. Moreover, the verifier should run U-1 times hash evaluations to generate the missing part of the Merkle proofs in the worst case (in particular for the first time in that verification epoch). The Merkle proof generation procedure of the verifier is only needed to run once in each verification epoch, and can run at idle time. Therefore, it has not much impact on the performance of password verification. Alternatively, RA could compute the whole sub-Merkle tree for the verifiers, but it may incur more communication overhead. Since U might not be big, we assume that the verifier has enough storage to cache the internal node of the Merkle tree. Such a cache strategy can facilitate the computation of the other Merkle proofs ($\mathrm{Pf}^{\mathrm{MTI}_i}_{vp^i_{\mathrm{ID}_j}}$), which is free in the average case. The verification of the Merkle proof associated with the password requires $\log^{U \cdot E}$ (i.e., the height of the Merkle tree gpk_G) times hash evaluations. Since the costs are dominated by the parameter U in the worst case, we fix $E = 630,720$ (for a six-year usage) in our measurements. We show the worst-case and average-case performance of DGTOne.Verify in Figure 11 (a) and (b), respectively. The password verification time is in the order of milliseconds (ms) which is efficient.

Identity Open Time. The procedure of DGTOne.Open is similar to DGTOne.Verify. However, it does not need to generate the Merkle proof anymore and has one more ASE decryption operation. The runtimes of it are shown in Figure 12.

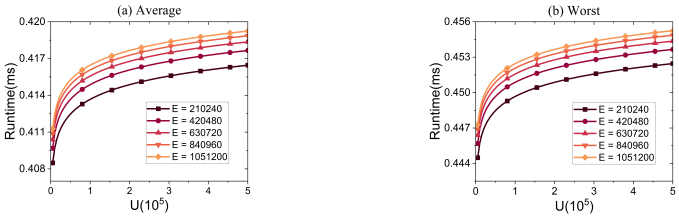

Fig. 12: Runtimes of Open.

Storage and Communication Costs. We consider the communication costs from the view of group members and verifiers, respectively. The communication overhead of a group member is determined by the size of each password, and it consists of 1 hash value with 256 bits, an identity ciphertext with 352 bits, and a 256-bit chameleon hash collision. It is 864 bits in total. The communication and storage costs of the verifier are mainly dominated by the sizes of a password and the group management messages downloaded from RA. In particular, the downloaded message consists of $U + \log^E$ hash values, U chameleon public keys (each of which has 520 bits), and U ciphertexts. Figure 13 shows the storage costs of RA and verifier, respectively.

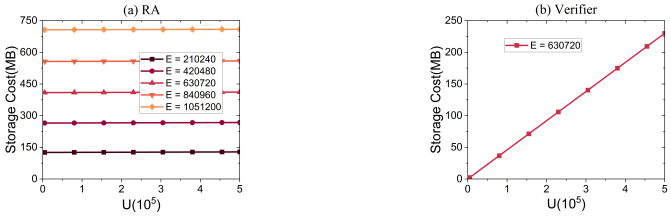

Fig. 13: Storage Costs of RA and Verifier.

We further evaluate the communication cost over an LTE network with 16Mbps bandwidth. For our first application, the TLS connection is implemented using *javax.net.ssl* to protect the communication between the corresponding entities (considering the aforementioned use cases). Approximately, the communication delays between the group member and the verifier are 0.45s (without TLS) and 0.9s (over TLS), respectively. The data transfer time between the RA and the verifier (with TLS) for $U = 5000$ and $E = 525600$ (as an example) is 1.25s. This delay is linear in U. However, such transmission is carried by each verifier only one time per verification epoch, and can be done at idle time.

Table 2: (D)GTOTP Comparison

Schemes	Group Setting		Execution Time									Storage Cost (Byte)			Password Size (Byte)
			RASetup	PInit	Revoke	Join	GetSD	PwGen	GMUpdate	Verify	Open	Member	Verifier	RA	
YJN+	Sta	Av	$2U \cdot E \cdot H + \phi \cdot Bfi$	$E(60H + 2AESe)$	-	-	2AESe	$30H$	-	$(32 + \log^{\frac{U \cdot E}{\phi}})H$ $+1BFc$	$(32 + \log^{\frac{U \cdot E}{\phi}})H$ $+1BFc + 1ASEd$	$1004E \cdot \log^{\frac{U \cdot E}{\phi}}$ $+16$	$1.44\epsilon \cdot \phi$	16	$76+$ $32\log^{\frac{U \cdot E}{\phi}}$
		Wo	$2U \cdot E \cdot H + \phi \cdot Bfi$	$E(60H + 2AESe)$	-	-	2AESe	$60H + 2AESe$	-	$(62 + \log^{\frac{U \cdot E}{\phi}})H$ $+1BFc$	$(62 + \log^{\frac{U \cdot E}{\phi}})H$ $+1BFc + 1ASEd$				
DGTOne	Dyn	Av	$(6U \cdot E + U + 1)AESe$ $+U \cdot E \cdot CHe + E \cdot PM$ $+(E \cdot U - 1)H + U \cdot E \cdot CHs$	$1AESs$	$1Ass$	$1AESe$	$2AESe$	$30H$	$(9U + 1)AESe + 1PM$ $+U(CHe + ASEe + CHs)$	$(32 + \log^{U \cdot E})H$ $+1CHe$	$(32 + \log^{U \cdot E})H + 1CHe$ $+1PM + 1ASEd + 3AESe$	36	$32 + 460U$ $+32log^{E}$	$16 + 32E \cdot log^{E}$ $+32E + 5U$	108
		Wo	$(6U \cdot E + U + 1)AESe$ $+U \cdot E \cdot CHe + E \cdot PM$ $+(E \cdot U - 1)H + U \cdot E \cdot CHs$	$1AESs$	$1Ass$	$1AESe$	$2AESe$	$62H + 10AESe$ $+1ASEe + 1CHe$	$(9U + 1)AESe + 1PM$ $+U(CHe + ASEe + CHs)$	$(62 + \log^{U \cdot E} + U)H$ $+1CHe$	$(62 + \log^{U \cdot E})H + 1CHe$ $+1PM + 1ASEd + 3AESe$				

6.2 Comparison

We mainly compare the group management setting and performance of our scheme with that of the YJN+ scheme [56] in the worst (Wo) case and the average (Av) case, respectively. We notice that YJN+ is the only scheme that allows us to make a relatively fair comparison because they satisfy similar security properties and functionalities. Both DGTOne and the YJN+ scheme provide anonymity and traceability, but DGTOne can satisfy them under a stronger dynamic group management setting.

We conduct the performance comparison based on the main operations of the corresponding algorithms. Here we compare the GVSTGen algorithm of the YJN+ scheme with DGTOne.RASetup since both of them aim to generate the group public key (which is known as the group verification state in the YJN+ scheme). The comparison results are shown in Table 2 (which also implies the results shown in the above figures). Meanwhile, we let 'BFc' and 'BFi' denote the costs of the check and the insert algorithms of Bloom filter, respectively. Moreover, we let 'ASEe' and 'ASEd' denote the encryption and decryption algorithms of AES-GCM-SIV, respectively. The cost of the setup and the encryption algorithms of AES is denoted with 'AESs' and 'AESe', respectively. We also let 'CHe', 'CHs', and 'CHc' denote the costs of the evaluation, setup, and collision algorithms of CH. The cost of permutation is denoted with 'PM'.

Although DGTOne can offer more complex dynamic (Dyn) group management than the static (Sta) group management in the YJN+ scheme, it does not introduce significantly higher overheads. In particular, for password generation, DGTOne incurs only a single additional chameleon hash collision evaluation (involving only some modular additions and multiplications) and a few more efficient symmetric cryptographic operations. As RA is powerful, the additional costs incurred are tolerable. The costs increased to the verifier are also not substantial. Moreover, our scheme has the advantage of shorter passwords sent by each group member. This is achieved by outsourcing the computation of the Merkle proof of a password to RA and the verifier, respectively.

7 Related Work

(Group) Time-based One-time Passwords. To overcome the weakness of server compromise threat, the recent time-based one-time passwords schemes prefer to adopt asymmetric key settings that can originate from Lamport construction [41]. One-time passwords (OTP) as an important authentication factor has a long research history that can date back to the seminal work by Lamport [41]. In recent years, Yu et al. [48] proposed a Time-based OTP authentication via Secure Tunnel (TOAST) to use the seed value stored on mobile phones to create OTP. Sun et al. [46] presented an OTP solution called TrustOTP to securely implement (existing) OTP schemes (e.g., S/Key [29] and HMAC-based TOTP [42]) over smartphones that can resist malicious mobile OS. Kogen et al. [39] revisited the Lamport scheme and proposed a variant with a tight security reduction in the random oracle model. Jin et al. [35] presented the new proof of the Lamport scheme in the Standard Model and innovatively used it to create a new cryptographic application called proof of aliveness.

However, the above-mentioned traditional TOTP schemes cannot provide anonymity since a party will only use her TOTP passwords at the server (verifier) where she registered. Erdem et al. [24] develop a cloud-based OTP service called OTPaaS for users. To achieve privacy, the users in OTPaaS need to register different pseudonyms and user-memorable passwords at the service provider and OTP provider. However, it cannot provide usage privacy against the OTP provider since the OTP provider needs to pair a user profile with a specific service provider. Besides, OTPaaS is only designed for single-sign-on applications without considering some

important security properties and functionalities of DGTOTP (i.e., group membership proofs, traceability, and public verifiability). In order to achieve lightweight anonymous authentication with group membership proofs, Yang et al. [56] recently generalized the asymmetric TOTP to a group setting to yield a new cryptographic notion, named GTOTP. However, the first GTOTP construction introduced by Yang et al. does not support dynamic group management, and suffers a large storage overhead at each group member.

Other Anonymous Authentication Techniques. In view of the importance of identity privacy protection in real life, it has always been valued by cryptographers. After decades of development, many related anonymous authentication technologies have been proposed. A classic technology to achieve identity privacy is to use pseudonyms [12,59,55] to anonymously identify entities, which are usually implemented based on the existing Public Key Infrastructure (PKI). That is, each entity has a long-term fixed identity certificate issued by a Certificate Authority (CA) used as its main certificate, and it also has a set of short-term identity certificates for temporary use as a pseudonym. By frequently replacing pseudonymous certificates, entities can avoid privacy threats such as long-term tracking and identity linking. However, common pseudonym technologies are not required to provide either group membership proofs or traceability, unlike DGTOTP.

For the client-server authentication scenario (i.e., without RA), anonymous password-based authenticated key exchange (APAKE) [51,54,49] is proposed to achieve anonymous-client session key establishment with user-memorable passwords. However, to realize client anonymity, these schemes leverage expensive public cryptographic primitives (such oblivious transfer [51,16], oblivious pseudo-random functions [30,49] and smooth projective hashing function [54,36]). Hence, APAKE protocols may not be efficient enough for resource-constrained devices. Also, they are not suitable for devices in Cyber-physical Systems that do not use user passwords. Moreover, the password verifier in APAKE has to be the server where the client is registered, so it cannot provide public verifiability. On the contrary, one of our motivations for DGTOTP is to achieve public verifiability, so that it can be used in many distributed applications (as presented in Section 5) in which the verifiers are post-determined and can be arbitrary entities.

Another important cryptographic technique related to anonymous authentication is Group Signature (GS) proposed by Chaum and Heyst [14], which allows a party to anonymously sign a message on behalf of a group. To study the provable security of group signatures, Bellare et al. [4] first defined a formal security model of static group signatures in 2003, which formally formulated many security threats faced by group signatures as two security properties: anonymity and traceability. In this paper, Bellare et al. proposed a generic group signature construction that can convert ordinary single-user digital signatures into group signatures through non-interactive zero-knowledge proof technology (NIZK). Many recent group signature algorithms (e.g. [6,2,8]) adopted a similar construction framework.

We note that some public key cryptography (PKC) based anonymous authentication technologies (such as pseudonym certificates and group signatures) can be used to implement DGTOTP. However, DGTOTP is proposed as a lightweight anonymous authentication primitive. The main construction challenge is the performance of the DGTOTP scheme on resource-constrained devices. Unfortunately, most PKC-based anonymous authentication technologies involve many expensive public key cryptographic operations, so they cannot be used efficiently on resource-constrained devices.

Table 3: Comparison of Password-based (Anonymous) Client Authentication.

	Password type	Security Properties			Group membership proof	Public Verifiable	Computation Overhead			Comm (Bytes)
		Anony	Trace	UPriv			Client	Verifier	RA	
[42]	Machine-usage	×	×	×	×	×	$2H+1TLS$	$2H+1TLS$	-	32
[48]	Machine-usage + User-Memorable	×	×	×	×	×	$2H+1TLS$	$2H+1TLS$	-	44
[24]	Machine-usage + User-Memorable	√	×	×	×	×	$2H+2TLS$	$2H+2TLS$	$2H+2TLS$	44
[58]	User-Memorable	√	×	×	×	×	$3GE+2mGE+1TLS$	$3GE+1mGE+1TLS$	-	8192
[57]	User-Memorable	√	×	×	×	√	$3ECM+2H+2TLS$	$6ECM+2H+2TLS$	$4ECM+2H+2TLS$	129
Ours	Machine-usage	√	√	√	√	√	$64H+10AESe+2ECM$ $+1ASEe+1CHc+1TLS$	$(64+\log^{U\cdot E}+U)H$ $+1CHe+2ECM+2TLS$	$(9U+1)AESe+1PM+1ECM$ $+U(CHe+ASEe+CHs)$	108

8 Discussions

Further Discussions on Privacy Issues. We do not consider privacy leakage caused by either the usage pattern of passwords or human behaviors in one verification epoch. In practice, a group member may run various applications using DGTOTP within one verification epoch. If two colluding verifiers belonging to different applications, who verify the passwords using the identical verification point, may link the behaviors of the corresponding group member across applications to learn extra privacy information, thus increasing the threats of breach of anonymity. Group members should also be aware of this kind of threat while using DGTOTP passwords. Such privacy leakage due to usage patterns also exists in other pseudonym schemes. To mitigate the privacy leakage within one verification epoch, a naive solution is to shorten the lifespan of verification epochs (say

one password per verification epoch). However, this solution may not be fit for the applications (e.g., distributed privacy-preserving proof of location [56]) that need multiple passwords of a verification epoch. Alternatively, a standard solution can be letting RA create multiple DGTOTP instances (which can run in parallel) for group members, so each group member can use disjoint DGTOTP instances in different applications. The group member can select an available DGTOTP instance for each application during the verification epoch without binding it to a specific application.

Comparison with (Anonymous) Client Authentication Schemes. We compare our client authentication scheme (i.e., the first use case shown in Section 5) with the password-based anonymous client authentication schemes [58,57]. In addition, we also list the performance of the standard HMAC-based TOTP scheme [42], and two (T)OTP-based authentication schemes TOAST [48] and OTPaaS [24] over secure channels for performance comparison. Meanwhile, we assume that the HMAC-based TOTP is used to instantiate the generic TOAST and OTPaaS frameworks in our comparison. We show the comparison results in Table 3. We assume each tuple stored at the AS is signed using ECDSA to provide integrity. We let GE and mGE denote the exponentiation and multi-exponentiation in a multiplicative group, respectively. And Let 'ECM' denote a point multiplication in ECC. We abuse TLS to denote the cost of establishing a secure channel using it. In the comparison, we mainly compare the security properties regarding anonymity and traceability, and the password usage privacy (UPriv) to RA (or IdP). We also compare the main performance on computation overheads for client authentication and client communication (Comm) over TLS. Password verifiability implies the usage scope in practice. The main difference among the compared protocols is the password type which determines the applications of the designed scheme. That is, our scheme is designed for the device of clients (with high entropy passwords). The other two constructions [58,57] are based on user-memorable passwords (with low entropy). So our scheme can be either used alone for machine authentication, or used as a second authentication factor to strengthen the user-centric client authentication schemes. Nevertheless, our scheme provides more security guarantees and has fewer client computation overheads. Although HMAC-based TOTP, TOAST, and OTPaaS are more efficient than DGTOTP, they cannot provide group membership proofs, traceability, and public verifiability as DGTOTP.

9 Conclusions

In this paper, we have defined the security notion for a dynamic group time-based one-time passwords scheme (DGTOTP) by extending the framework of static GTOTP [56]. An efficient DGTOTP is proposed to satisfy the security of our defined model without random oracles. Our proposal can achieve constant-sized storage for group members (via an elaborately tailored outsourcing approach) compared to the previous static GTOTP scheme. We also presented a secure DGTOTP password application approach with low latency based on the context of the traditional client authentication using passwords over TLS, that can particularly resist password replay attacks in an arbitrary-verifier setting. This can make DGTOTP more practical in the real world. However, since DGTOTP passwords are publicly verifiable, they can be used as a generic tool to build other distributed privacy-preserving application protocols. That is, the verifier in such an application can be any entities beyond a web server. Nevertheless, we encourage readers to explore more applications of DGTOTP.

As DGTOTP is in its early stages of development, there are still many open questions. For example, it might be interesting to construct a DGTOTP scheme with stronger security (such as membership privacy [2] and non-frameability [8]). It is also worthwhile to devise a DGTOTP solution with constant communication costs at the verifiers. Another open challenge is to optimize the maximum number of members that can join, particularly for a long life span, while ensuring efficiency.

Acknowledgement

This work was supported in part by the Natural Science Foundation of China under Grant 62372386, Grant 62032019, Grant 62032005, and Grant 61972094; and in part by the Natural Science Foundation of Chongqing under Grant CSTB2022NSCQ-MSX0437.

References

1. Source codes of dgtotp. https://github.com/I123T/DGTOTP, accessed: 2022-10-1
2. Backes, M., Hanzlik, L., Schneider-Bensch, J.: Membership privacy for fully dynamic group signatures. In: CCS. pp. 2181–2198. ACM (2019)
3. Barker, E., Barker, W.C.: Recommendation for Key Management: Part 2 – Best Practices for Key Management Organizations. US Department of Commerce, Technology Administration, National Institute of Standards and Technology (2019)
4. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: EUROCRYPT. pp. 614–629. Springer (2003)

5. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: EUROCRYPT. pp. 409–426. Springer (2006)
6. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: CT-RSA. pp. 136–153. Springer (2005)
7. Boeira, F., Asplund, M., Barcellos, M.P.: Vouch: A secure proof-of-location scheme for vanets. In: MSWiM. pp. 241–248. ACM (2018)
8. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J.: Foundations of fully dynamic group signatures. J. Cryptol. **33**(4), 1822–1870 (2020)
9. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN. pp. 163–169. Springer (1998)
10. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM **51**(4), 557–594 (2004)
11. Cao, X., Yang, Z., Ning, J., Jin, C., Lu, R., Liu, Z., Zhou, J.: Dynamic group time-based one-time passwords. IEEE Trans. Inf. Forensics Secur. **19**, 4897–4913 (2024)
12. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981)
13. Chaum, D.: Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In: EUROCRYPT. pp. 241–244. Springer (1985)
14. Chaum, D., van Heyst, E.: Group signatures. In: EUROCRYPT. pp. 257–265. Springer (1991)
15. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: CCS. pp. 719–728. ACM (2017)
16. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: CRYPTO. pp. 502–534. Springer (2021)
17. Culnane, C., Schneider, S.A.: A peered bulletin board for robust use in verifiable voting systems. In: CSF. pp. 169–183. IEEE (2014)
18. Dierks, T., Rescorla, E.: The transport layer security (tls) protocol version 1.2. Tech. rep. (2008)
19. Dong, X., Sun, S., Shi, D., Gao, F., Wang, X., Hu, L.: Quantum collision attacks on aes-like hashing with low quantum random access memories. In: ASIACRYPT. pp. 727–757. Springer (2020)
20. Donlan, B., ForClojureGuy.: Legion of the bouncy castle inc. bc-fja 1.0.2 (bouncy castle fips java api). https://downloads.bouncycastle.org/fips-java/BC-FJA-UserGuide-1.0.2.pdf (2022)
21. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: CCS. pp. 1197–1210. ACM (2015)
22. Duo: Duo security. https://duo.com/ (2018), Accessed: Dec. 2018
23. Emura, K., Hayashi, T., Ishida, A.: Group signatures with time-bound keys revisited: A new model, an efficient construction, and its implementation. IEEE Trans. Dependable Secur. Comput. **17**(2), 292–305 (2020)
24. Erdem, E., Sandikkaya, M.T.: Otpaas - one time password as a service. IEEE Trans. Inf. Forensics Secur. **14**(3), 743–756 (2019)
25. Gajek, S., Manulis, M., Sadeghi, A., Schwenk, J.: Provably secure browser-based user-aware mutual authentication over TLS. In: AsiaCCS. pp. 300–311. ACM (2008)
26. Garman, C., Paterson, K.G., van der Merwe, T.: Attacks only get better: Password recovery attacks against RC4 in TLS. In: Security. pp. 113–128. USENIX (2015)
27. Google: Google 2-step verification. https://www.google.com/landing/2step/ (2018), Accessed: Dec. 2018
28. Gueron, S.: Aes-gcm-siv implementations (128 and 256 bit) (2018), https://github.com/Shay-Gueron/AES-GCM-SIV
29. Haller, N.: The s/key one-time password system (1995)
30. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. J. Cryptol. **23**(3), 422–456 (2010)
31. Hosoyamada, A., Sasaki, Y.: Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In: EUROCRYPT. pp. 249–279. Springer (2020)
32. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: Authenticated confidential channel establishment and the security of TLS-DHE. J. Cryptol. **30**(4), 1276–1324 (2017)
33. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: EUROCRYPT. pp. 456–486. Springer (2018)
34. Ji, S., Yang, S., Wang, T., Liu, C., Lee, W., Beyah, R.A.: PARS: A uniform and open-source password analysis and research system. In: ACSAC. pp. 321–330. ACM (2015)
35. Jin, C., Yang, Z., van Dijk, M., Zhou, J.: Proof of aliveness. In: ACSAC. pp. 1–16. ACM (2019)
36. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: ASIACRYPT. pp. 636–652. Springer (2009)
37. Khodaei, M., Noroozi, H., Papadimitratos, P.: Scaling pseudonymous authentication for large mobile systems. In: WiSec. pp. 174–184. ACM (2019)
38. Kiayias, A., Kuldmaa, A., Lipmaa, H., Siim, J., Zacharias, T.: On the security properties of e-voting bulletin boards. In: SCN. pp. 505–523. Springer (2018)
39. Kogan, D., Manohar, N., Boneh, D.: T/key: Second-factor authentication from secure hash chains. In: CCS. pp. 983–999. ACM (2017)
40. Krawczyk, H., Rabin, T.: Chameleon signatures. In: NDSS. The Internet Society (2000)
41. Lamport, L.: Password authentication with insecure communication. Commun. ACM **24**(11), 770–772 (1981)
42. M'Raihi, D., Machani, S., Pei, M., Rydell, J.: Totp: Time-based one-time password algorithm (2011), https://tools.ietf.org/html/rfc6238
43. Rescorla, E.: The transport layer security (tls) protocol version 1.3. Tech. rep. (2018)

44. Rivest, R.L., Weitzner, D., Ivers, L., Soibelman, I., Zissman, M.: Pact: Private automated contact tracing (2020)
45. Schäge, S., Schwenk, J., Lauer, S.: Privacy-Preserving Authenticated Key Exchange and the Case of IKEv2. In: PKC. pp. 567–596. Springer (2020)
46. Sun, H., Sun, K., Wang, Y., Jing, J.: Trustotp: Transforming smartphones into secure one-time password tokens. In: CCS. pp. 976–988. ACM (2015)
47. Troncoso, C., Payer, M., Hubaux, J.P., Salathé, M., Larus, J., Bugnion, E., Lueks, W., Stadler, T., Pyrgelis, A., Antonioli, D., et al.: Decentralized privacy-preserving proximity tracing. Data Eng. Bull. **43**(2), 36–66 (2020)
48. Uymatiao, M.L.T., Yu, W.E.S.: Time-based otp authentication via secure tunnel (toast): A mobile totp scheme using tls seed exchange and encrypted offline keystore. In: ICIST. pp. 225–229. IEEE (2014)
49. Vasco, M.I.G., del Pozo, A.L.P., Soriente, C.: A key for john doe: Modeling and designing anonymous password-authenticated key exchange protocols. IEEE Trans. Dependable Secur. Comput. **18**(3), 1336–1353 (2021)
50. Verheul, E., Hicks, C., Garcia, F.D.: IFAL: issue first activate later certificates for V2X. In: EuroS&P. pp. 279–293. IEEE (2019)
51. Viet, D., Yamamura, A., Tanaka, H.: Anonymous password-based authenticated key exchange. In: INDOCRYPT. pp. 244–257. Springer (2005)
52. Wang, D., Wang, P.: Two birds with one stone: Two-factor authentication with security beyond conventional bound. IEEE Trans. Dependable Secur. Comput. **15**(4), 708–722 (2018)
53. Waters, B.R., Felten, E.W.: Secure, private proofs of location. Tech. rep. (2002), www.cs.princeton.edu/research/techreps/TR-667-03
54. Yang, X., Jiang, H., Xu, Q., Hou, M., Wei, X., Zhao, M., Choo, K.R.: A provably-secure and efficient verifier-based anonymous password-authenticated key exchange protocol. In: Trustcom/BigDataSE/ISPA. pp. 670–677. IEEE (2016)
55. Yang, Z., Dinh, T.T.A., Yin, C., Yao, Y., Yang, D., Chang, X., Zhou, J.: LARP: A lightweight auto-refreshing pseudonym protocol for V2X. In: SACMAT. pp. 49–60. ACM (2022)
56. Yang, Z., Jin, C., Ning, J., Li, Z., Dinh, A., Zhou, J.: Group time-based one-time passwords and its application to efficient privacy-preserving proof of location. In: ACSAC. pp. 497–512. ACM (2021)
57. Zhang, Z., Wang, Y., Yang, K.: Strong authentication without temper-resistant hardware and application to federated identities. In: NDSS (2020)
58. Zhang, Z., Yang, K., Hu, X., Wang, Y.: Practical anonymous password authentication and TLS with anonymous client authentication. In: CCS. pp. 1179–1191. ACM (2016)
59. Zimmer, E., Burkert, C., Petersen, T., Federrath, H.: PEEPLL: privacy-enhanced event pseudonymisation with limited linkability. In: SAC. pp. 1308–1311. ACM (2020)

A Lists of Notions

We list the main notations used in this paper in Table 4.

Table 4: Some Important Notations.

G	Group identity
ID, α_{ID}	The original identity and transformed identity (in terms of join time) of a group member
T_s, T_e	Start and end times of a protocol instance, respectively
Δ_e, Δ_s	Life-spans of verification point and password, respectively.
U, E, N	Numbers of group members, verification epochs, and passwords (in one verification epoch), respectively.
vp_{ID}	Verification point of a group member ID
$\mathsf{gpk}_{\mathsf{G}}$	Group public key
sd^i_{ID}	The i-th secret seed for generating the i-the verify-point of ID.
sk_{ID}	Secret key for generating the secret seeds of ID.
$pw^{i,z}_{\mathsf{ID}}$	The z-th password of ID in the i-th verify-epoch.
T_i, T_{ct}	The i-th time slot and the current system time slot, respectively.
GM	Group management message
MPI	A list of public keys of chameleon hash and identity ciphertexts
IDL	A list of registered group members' identities
RL	Revocation list

B A DGTOTP Variant with Cache

If each group member has enough storage space, DGTOne can also be straightforwardly modified to let each group member cache the Merkle proofs, and chameleon collisions and public key pairs during the join procedure.

That is, a group member ID_j can generate and include all her verification points in $\mathsf{vst}_{\mathsf{ID}_j} = \{vp^i_{\mathsf{ID}_j}\}_{i\in[E]}$ in the PInit algorithm. So that RA can compute all chameleon collisions, and append the corresponding Merkle proofs, and chameleon hash collisions and public key pairs into $\mathsf{Ax}_{\mathsf{ID}_j}$ which will be stored by ID_j locally. As a result, the storage cost of each group member has a magnitude in $O(E \cdot \log^U)$. Consider a scenario with $U = 5,000$ (for a small group) and $E = 105,120$ (for one-year usage under $\Delta_e = 5$min), the size of $\mathrm{DST}_{\mathsf{G}}$ in GM_{G} is around 108.5MB (with the implementation parameters in Section 6), which might be also acceptable.

Another storage optimization strategy that each group member could adopt is to cache the Merkle proofs segmented by a fixed number of verification epochs (e.g., one week). In this way, each group member can periodically submit the verification points used for the next segment to RA and receive back the corresponding Merkle proofs and chameleon hash public keys. As after the registration is done, RA and a group member can share a symmetric key which can used to derive a key for generating message authentication code to authenticate the subsequent verification points in other segments. Hopefully, such a poof-segmented caching strategy can reduce the storage cost of each group member to be less than 1MB. We stress that this strategy is different from letting a group member join many times, since each join operation would consume a transformed identity (This may affect the size of each sub-Merkle tree).

C Proof of Theorem 1

We present proof with the following sequence of games starting from the real Anony game. We will gradually change the games one after another until the advantage of the adversary in a game is vanished. Meanwhile, we let $\mathsf{BK}^{\mathsf{Anony}}_i$ be an event that the Anony game returns 1 in the i-th game.

Game 0. This game is the real Anony game. We have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_0] = 1/2 + \mathsf{Adv}^{\mathsf{Anony}}_{\mathcal{A},\mathsf{DGTOne}}(\kappa, \mathsf{setpms}).$$

Game 1. This game proceeds exactly like the previous game, but the challenger aborts if two $\mathsf{F}_2.\mathsf{Setup}$ executions output the same key. This abort event would occur with negligible probability, otherwise we can break the security of PRF. Consider the case that the PRF challenger runs $\mathsf{F}_2.\mathsf{Setup}$ to get a challenge key k^*. $\mathcal{F}$ can run $\mathsf{F}_2.\mathsf{Setup}$ algorithms herself $U-1$ times to get PRF keys $k_1, \ldots, k_{U-1}$. Then, the abort event implies that there are two identical keys (k'_1, k'_2) in the set $(k^*, k_1, \ldots, k_{U-1})$. With a probability $2/U$, either k'_1 or k'_2 equals to k^*. If so, the adversary knows the k^* that can enable her to break the security of PRF. Due to the security of F_2, we have

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_0] \leq \Pr[\mathsf{BK}^{\mathsf{Anony}}_1] + \frac{U}{2} \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}_2}(\kappa, 1).$$

Game 2. We change this game from the previous game by adding one abort condition. Namely, the challenger aborts if it fails to guess which two honest group members' TOTP instances that are involved in the Anony $\mathsf{Challenge}$ query. Specifically, the challenger needs to guess correctly not only the transformed identities $\alpha_{\hat{\mathsf{ID}}_0}$ and $\alpha_{\hat{\mathsf{ID}}_1}$ (from at most U group members) but also the verification epoch $i^* \in [E]$ (from E verification epochs), which will be chosen in the Anony $\mathsf{Challenge}$ query. The probability of a correct guess is bound to $1/2U \cdot E$. Hence, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_1] = 2U \cdot E \cdot \Pr[\mathsf{BK}^{\mathsf{Anony}}_2].$$

Game 3. In this game, we change the PRF-derived secrets (from RA's secret seed k_{RA}) in the set $\{ks_{\alpha_{\hat{\mathsf{ID}}_0}}, ks_{\alpha_{\hat{\mathsf{ID}}_1}}\}$ to truly random values. We claim that if there exists an adversary that can distinguish this game from the previous game then she must be able to break the PRF security of F_1. To prove this claim, we can further play three sub-games $\{G2.0, G2.1, G2.2\}$, in which $G2.0$ is identical to Game 2. In $G2.1$ and $G2.2$, we change $\{ks_{\alpha_{\hat{\mathsf{ID}}_0}}, ks_{\alpha_{\hat{\mathsf{ID}}_1}}\}$ to random values, respectively. So $G2.2$ is just identical to this game. Any adversary $\mathcal{D}$ that can distinguish between $G2.0$ and $G2.1$ can be used to build an efficient algorithm $\mathcal{B}$ to break the PRF security of F_1. $\mathcal{B}$ can simulate the Anony game for $\mathcal{D}$. As for the generation of kr, $\mathcal{B}$ can query $\mathsf{Challenge}(\mathsf{G}||\text{“KS”}||\alpha_{\hat{\mathsf{ID}}_0})$ to get r^*_b and return it to $\mathcal{D}$. Meanwhile, $\mathcal{B}$ can simulate the rest of the values using the secret keys of her own choice. If r^*_b is a result of the F_1 then the simulation is identical to $G2.0$, otherwise the game is the same as $G2.1$. $\mathcal{B}$ can forward the distinguishing result of $\mathcal{D}$ to its PRF challenger to win the PRF game by our assumption. Similarly, we can generalize this argument to any game pair $G2.x$ and $G2.(x-1)$ (for $1 \leq x \leq 2$), which can prove the claim of this game. Since there are PRF-derived secrets, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_2] \leq \Pr[\mathsf{BK}^{\mathsf{Anony}}_3] + 3 \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}_1}(\kappa, 1).$$

Game 4. The game proceeds exactly as the previous game, but we further change the outputs of F_1 in the set $\{ke^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}}, ke^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}}, re^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}}, re^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}}, dvp^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}}, dvp^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}}, rd^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}}, rd^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}}, rk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}}, rk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}}\}$ with truly random values, where i^* denotes the guessed challenge verification epoch. With the similar arguments in the Game 3, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_3] \leq \Pr[\mathsf{BK}^{\mathrm{Anony}}_4] + 10 \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}_1}(\kappa, E).$$

The modifications in this game imply that all those values (including the chameleon keys) are sampled randomly to be independent of the secret seed of the corresponding group members.

Game 5. We change this game from the previous game by using the real verification points $(\hat{vp}^{i^*}_{\hat{\mathsf{ID}}_0}, \hat{vp}^{i^*}_{\hat{\mathsf{ID}}_1})$ and two random values $(r^{i}_{\hat{\mathsf{ID}}_0}, r^{i^*}_{\hat{\mathsf{ID}}_1})$ to compute their assigned Merkle tree leaves via CH (instead of computing the chameleon collision). Since the chameleon secret keys, and dummy verification point and randomness pairs are replaced with truly random values. The random values (r^*_0, r^*_1) are statistically close with a negligible distance $\mathsf{negl}(\kappa)$ to the chameleon collisions derived following the specification of DGTOne. Namely, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_4] \leq \Pr[\mathsf{BK}^{\mathsf{Anony}}_5] + \mathsf{negl}(\kappa).$$

For all other chameleon collisions (including those from malicious group members joined via AddMM queries), the challenger can use its secret key to compute the collision as in the original DGTOne (as all chameleon hash keys are independently generated).

Game 6. In this game, we let the challenger abort if the dummy verification point and randomness pairs in the set $\{dvp^{i^*}_{\hat{\mathsf{ID}}_j}, rd^{i^*}_{\hat{\mathsf{ID}}_j}\}_{0\leq j\leq 1}$ result in the same chameleon hash value. Obviously, if such an abort event occurs with non-negligible probability, then it can be used to break the security of CH. As the randomness for generating the chameleon keys are replaced with random values, we can set $pk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}}$ as a chameleon-challenge key pk^*. The other chameleon key pairs $(sk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}}, pk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}})$ are generated as before. By assumption, we have that $sk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}} \cdot rd^{i^*}_{\hat{\mathsf{ID}}_1} + dvp^{i^*}_{\hat{\mathsf{ID}}_1} = sk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}} \cdot rd^{i^*}_{\hat{\mathsf{ID}}_0} + dvp^{i^*}_{\hat{\mathsf{ID}}_0}$. Then, we can compute $sk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_0}} := \frac{sk^{i^*}_{\alpha_{\hat{\mathsf{ID}}_1}} \cdot rd^{i^*}_{\hat{\mathsf{ID}}_1} + dvp^{i^*}_{\hat{\mathsf{ID}}_1} - dvp^{i^*}_{\hat{\mathsf{ID}}_0}}{rd^{i^*}_{\hat{\mathsf{ID}}_0}}$. Namely, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_5] \leq \Pr[\mathsf{BK}^{\mathsf{Anony}}_6] + \mathsf{Adv}^{\mathsf{CHS}}_{\mathcal{A},\mathsf{CH}}(\kappa).$$

Game 7. The game proceeds exactly as the previous game, but the challenge always uses $\hat{\mathsf{ID}}_0$ to generate the ciphertext used by the $\mathsf{Anony\ Challenge}$ query without considering the challenge-bit b. If there exists an adversary $\mathcal{A}$ who can distinguish this game from the previous game then we can make use of it to build an algorithm $\mathcal{C}$ to break the security of ASE. Specifically, $\mathcal{C}$ can query the identities $(\alpha_{\hat{\mathsf{ID}}_0}, \alpha_{\hat{\mathsf{ID}}_1})$ to the ASE challenger and use the result to simulate the challenge ciphertext used in the $\mathsf{Anony\ Challenge}$ query. All other identity-ciphertexts can be obtained by calling an encryption oracle simulated by the ASE challenger. Let bc be the bit sampled by the ASE challenger. Note that if $bc = 0$, the simulated game is identical to the previous game, otherwise it equals to this game. Thus, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_6] \leq \Pr[\mathsf{BK}^{\mathrm{Anony}}_7] + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{ASE}}(\kappa, E).$$

The changed game implies that the adversary cannot infer any information regarding the identity from the ASE ciphertext.

Game 8. This game proceeds as before but the challenger randomly chooses a leaf from the two leaves that are meant to use for $\hat{\mathsf{ID}}_0$ and $\hat{\mathsf{ID}}_1$ without using the result of the permeation scheme. If there exists a PPT adversary $\mathcal{A}$ that can distinguish this game from the previous game, then we build an efficient algorithm $\mathcal{B}$ using $\mathcal{A}$ to break the security of PM. During the simulation for $\mathcal{A}$, $\mathcal{B}$ can query a PM-challenge query with input $(\{1, \ldots, U\}, w_0, w_1)$, where w_0 and w_1 refer to the joined orders of $\hat{\mathsf{ID}}_0$ and $\hat{\mathsf{ID}}_1$, respectively. The leaf chosen by $\mathcal{B}$ has a probability $1/2$ to be failing to match the output of PM, then the game is equal to this game. In this case, $\mathcal{A}$'s output bit b' (we assume $b' = 0$ denotes the previous game) can be used to help $\mathcal{B}$ to output the right y_{β_0} with a non-negligible probability by assumption. Namely, if $b' = 0$ $\mathcal{B}$ submits the index of the leaf chosen by itself to $\mathsf{PM\ Finalize}$; otherwise, it submits the other one. Thus, we have that

$$\Pr[\mathsf{BK}^{\mathsf{Anony}}_7] \leq \Pr[\mathsf{BK}^{\mathrm{Anony}}_8] + \mathsf{Adv}^{\mathsf{UP}}_{\mathcal{A},\mathsf{PM}}(\kappa).$$

Finally, we claim that the leaf positions assigned to verification points do not leak any information regarding their owners. For the extreme case that all group members except for $\hat{\mathsf{ID}}_0$ and $\hat{\mathsf{ID}}_1$ are malicious and created by adversary via the AddMM procedure, there are still at least two leaf positions which are randomly assigned to the verification points of $\hat{\mathsf{ID}}_0$ and $\hat{\mathsf{ID}}_1$. Namely, the probability that a leaf is bound to a verification point is at least $1/2$. In other words, the adversary cannot gain any non-negligible advantage over $1/2$ in this game, so that we have $\Pr[\mathsf{BK}^{\mathrm{Anony}}_8] = 1/2$.

Putting together the advantages in all the above games can only result in negligible advantage, which proves Theorem 1.

D Proof of Theorem 2

We denote with $\mathsf{BK}^{\mathsf{Trace}}_i$ an event that the i-th (modified) Trace game outputs 1.

Game 0. This game is original Trace game. I.e., we have that

$$\Pr[\mathsf{BK}_0^{\mathsf{Trace}}] = \mathsf{Adv}^{\mathsf{Trace}}_{\mathcal{A},\mathsf{DGTOne}}(\kappa, \mathsf{setpms}).$$

Game 1. This game proceeds as the same as the previous game but the challenger aborts if two honest group members have the same PRF keys. By applying the similar argument in the Game 1 of the proof of Theorem 1, we have that

$$\Pr[\mathsf{BK}_0^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_1^{\mathsf{Trace}}] + \frac{U}{2} \cdot \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}_2}(\kappa, 1).$$

Game 2. Let vp^* be the verification point derived from the pw^* (provided by the adversary in the Finalize query. We change this game from the previous game by adding an abort rule to let challenger abort if the following conditions are satisfied: i) $\mathsf{H}_1(vp^*||C^*_{\mathsf{ID}_j}||i^*) = \mathsf{H}_1(vp^u_{\mathsf{ID}_j}||C^u_{\mathsf{ID}_j}||u)$; ii) and ID^* is registered by AddHM, where i^* and u can be arbitrary indices. The adversary which can make such an abort event with non-negligible probability can be used to break the security of H_1. Thus, we have that

$$\Pr[\mathsf{BK}_1^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_2^{\mathsf{Trace}}] + \mathsf{Adv}^{\mathsf{CRH}}_{\mathcal{A},\mathsf{H}_1}(\kappa).$$

Game 3. This game proceeds exactly as the previous game but the challenger aborts if $\mathcal{A}$ outputs a password $pw^* = (\bar{pw}^{*i,z}_{\mathsf{ID}_j}, r^i_{\mathsf{ID}^*}, C^i_{\mathsf{ID}^*}, \mathrm{Pf}^i_{\mathsf{ID}^*})$ for T^* s.t.: i) $\mathsf{Verify}(\mathsf{gpk}_\mathsf{G}, \bar{pw}^{*i,z}_{\mathsf{ID}_j}, T^*, \mathrm{RL}_\mathsf{G}) = 1$; ii) $\mathrm{Pf}^i_{\mathsf{ID}^*}$ is not computed by the challenger (in either AddHM or AddMM procedures). If the adversary can lead to the abort event occurring with a non-negligible probability, then we can make use of $\mathcal{A}$ to break the security of the Merkle tree scheme. Thus, we have that

$$\Pr[\mathsf{BK}_2^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_3^{\mathsf{Trace}}] + \mathsf{Adv}^{\mathsf{MT_Forge}}_{\mathcal{A},\mathsf{MT}}(\kappa).$$

Game 4. The challenger proceeds with this game as before but aborts if it fails to guess the owner and the verification epoch of the forged password of the adversary. The overall probability of a correct guess is bound by $\frac{1}{U \cdot E}$. Thus, we have that $\Pr[\mathsf{BK}_3^{\mathsf{Trace}}] = U \cdot E \cdot \Pr[\mathsf{BK}_4^{\mathsf{Trace}}]$. We particularly denote the guessed verification epoch by i^*-th and the target uncorrupted group member by ID^*, respectively.

Game 5. The game proceeds exactly as the previous game, but we further change the outputs of F_1 in the set $\{ke^{i^*}_{\alpha_{\mathsf{ID}^*}}, re^{i^*}_{\alpha_{\mathsf{ID}^*}}, dvp^{i^*}_{\alpha_{\mathsf{ID}^*}}, rd^{i^*}_{\alpha_{\mathsf{ID}^*}}, rk^{i^*}_{\alpha_{\mathsf{ID}^*}}\}$ with truly random values, where i^* denotes the guessed challenge verification epoch. With similar arguments in the Game 3 of the proof of Theorem 1, we have that

$$\Pr[\mathsf{BK}_4^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_5^{\mathsf{Trace}}] + 5\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},\mathsf{F}_1}(\kappa, E).$$

Game 6. We change this game from the previous game by using the real verification point $\hat{vp}^i_{\mathsf{ID}^*}$ and random value $r^i_{\mathsf{ID}^*}$ to compute its assigned Merkle tree leaves via CH. This change is just conceptual since the random value $r^i_{\mathsf{ID}^*}$ is statistically close to the collision generated as the original scheme. Namely, we have that

$$\Pr[\mathsf{BK}_5^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_6^{\mathsf{Trace}}] + \mathsf{negl}(\kappa).$$

Game 7. In this game, we let the challenger abort if the adversary submit a password pw^* resulting in a $(vp^{i^*}_{\mathsf{ID}_\mathcal{A}}, r^{i^*}_{\mathsf{ID}_\mathcal{A}}\})$ s.t. $\mathsf{CH.Eval}(pk^{i^*}_{\alpha_{\mathsf{ID}^*}}, vp^{i^*}_{\mathsf{ID}_\mathcal{A}}, r^{i^*}_{\mathsf{ID}_\mathcal{A}}) = \mathsf{CH.Eval}(pk^{i^*}_{\alpha_{\mathsf{ID}^*}}, vp^{i^*}_{\mathsf{ID}^*}, r^i_{\mathsf{ID}^*})$. If such an abort event occurs with non-negligible probability, then it can be used to break the security of CH. I.e., the collision event can be exploited to extract the chameleon secret key. Namely, we have that

$$\Pr[\mathsf{BK}_6^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_7^{\mathsf{Trace}}] + \mathsf{Adv}^{\mathsf{CHS}}_{\mathcal{A},\mathsf{CH}}(\kappa).$$

As a result, the adversary cannot bind a Merkle tree leaf with verification point and identity ciphertext chosen by herself.

Game 8. The challenger proceeds with this game as before as before, but aborts if an adversary $\mathcal{A}$ outputs a valid password $\bar{pw}^*$ of ID^* at some time T^* such that T^* is greater than all previously opened password values. Obviously, such a tuple $(\bar{pw}^*, T^*)$ can be used to break the TOTP security. We can build an algorithm $\mathcal{B}$ by running $\mathcal{A}$ as a sub routine and simulate the modified Trace game (as the previous game). $\mathcal{B}$ can appropriately query $\mathsf{GetNextPw}()$ from the TOTP challenger to simulate the passwords of ID^* before time T^*. By applying the security of TOTP, we have that

$$\Pr[\mathsf{BK}_7^{\mathsf{Trace}}] \leq \Pr[\mathsf{BK}_8^{\mathsf{Trace}}] + \mathsf{Adv}^{\mathsf{TOTP_Forge}}_{\mathcal{A},\mathsf{TOTP}}(\kappa, T_s, T_e, \Delta_s).$$

The advantage of this game is 0 since the adversary cannot manipulate the elements in any passwords of uncorrupted group members in this game. This concludes the proof.

E Security Enhancement for ACACCE

The formal security definition of anonymous client authentication over TLS can be found in [58]. Informally speaking, an anonymous client authentication protocol should prevent any PPT adversary from either impersonating an honest (uncorrupted) group member in a group G to the server or inferring the identity of an honest group member from the protocol transcript (We refer the readers to [58] for more details).[10]

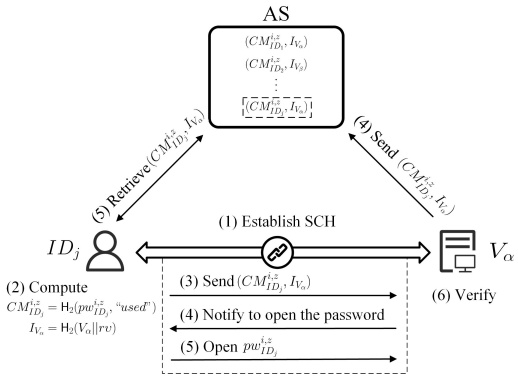

Fig. 14: Overview of Anonymous Client Authentication with DGTOTP.

We assume that a semi-honest authentication server (AS) exists to sign all stored elements with its private key. We specifically describe the main steps of our solution below (See also in Figure 14):

1. A group member ID_j first establishes a S-ACCE secure channel $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$ with an arbitrary verifier V_a;
2. ID_j computes a commitment $CM^{i,z}_{\mathsf{ID}_j} := \mathsf{H}_2(pw^{i,z}_{\mathsf{ID}_j}, \text{“used”})$ as a password-usage token, samples a randomness $rv \xleftarrow{\$} \{0,1\}^*$ and computes a randomized identifier of the verifier $I_{\mathsf{V}_a} := \mathsf{H}_2(\mathsf{V}_a||rv)$;
3. ID_j sends $(CM^{i,z}_{\mathsf{ID}_j}, I_{\mathsf{V}_a}, rv)$ to V_a over $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$, i.e., the tuple is encrypted by an ASE scheme and the resulting ciphertext is protected by a MAC scheme based on the session key of $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$;
4. V_a checks whether $CM^{i,z}_{\mathsf{ID}_j}$ has been recorded at the AS and $I_{\mathsf{V}_a} = \mathsf{H}_2(\mathsf{V}_a||rv)$. If so, it rejects in $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$ immediately. Otherwise, it stores the tuple $(CM^{i,z}_{\mathsf{ID}_j}, I_{\mathsf{V}_a})$ at the AS and notifies ID_j to open the password;
5. Upon receiving the open notification from $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$, ID_j queries the tuple $(CM^{i,z}_{\mathsf{ID}_j}, I_{\mathsf{V}_a})$ to the AS. If such a tuple exists at the AS, then ID_j opens the $pw^{i,z}_{\mathsf{ID}_j}$ to V_a over $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$;
6. Finally, V_a verifies whether $CM^{i,z}_{\mathsf{ID}_j} = \mathsf{H}_2(pw^{i,z}_{\mathsf{ID}_j}, \text{“used”})$ to complete the client authentication.

The performance of the above scheme at group members would be much more efficient than the previous anonymous client authentication (e.g., [57] using NIZK (involving many exponentiations). Our solution only needs a few hash function evaluations to generate the DGTOTP passwords and some operations (e.g., regular signature verification) for verifying the integrity of the tuple $(CM^{i,z}_{\mathsf{ID}_j}, I_{\mathsf{V}_a})$ stored at the AS. However, unlike [57], we do not need a trustworthy identity provider (IdP) to online generate the assertions for clients (that may leak the usage privacy to the IdP). Since the stored data at AS are random values, both RA and AS in our scheme do not know the password usage status (e.g., usage time and communication partners) of group members. We further summarize comparison results in Appendix 8.

For intuition and simplicity, we only illustrate our security enhancement based on S-ACCE. It might be interesting to extend the above idea to build a mutual anonymous authenticated credential channel establishment scheme using DGTOTP passwords of both parties. That is, it is needed to tailor the messages stored at the AS to ensure the one-time usage of each password. We leave the detailed construction as future work.

In practice, the AS can also be realized by public bulletin board (PBB) [17,15,38]. Some volunteers (say group members) can monitor the historical data of the PBB to detect malicious behaviors of it. Of course, some of the existing PBBs (e.g., [17,15,38]) can be slightly weakened (by allowing short-time storage of data) to meet the above requirements.

Security Analysis. We assume that the AS provides time-based integrity, the S-ACCE protocol and the DGTOTP scheme are secure, and H_2 can be modeled as a random oracle, then the DGTOTP password used

[10] Zhang et al. [58] defines the anonymous client authentication in authenticated key exchange (AKE). As discussed in [32,45], an AKE security model can be transformed into an ACCE security model with slight modifications (i.e., change the formulation of session key security to the security of data sent over the established credential channel). Since the DGTOTP passwords are the data sent over the TLS channel, so we prefer to use the security notion of ACCE.

in the above scheme can resist the *password replay attacks* and provide anonymous client authentication. It is straightforward to see that the client anonymity is realized by DGTOne. So we focus on the security analysis against password replay attacks. We can slightly modify our DGTOTP model to formulate the password replay attacks. Specifically, we can change the $\mathsf{ReceivePw}(pw)$ procedure to use a variable cnt_{pw} to record the number of times that a password pw has passed the verification. We add another condition in $\mathsf{Finalize}(b^*, pw^*, T^*)$ for outputting 1 if $cnt_{pw^*} > 1$.

As the DGTOTP scheme is secure, no PPT adversary can query H_2 with an unused password of an honest group member. The tuple $(CM^{i,z}_{\mathsf{ID}_j}, I_{\mathsf{V}_a})$ means that the password will be verified by V_a. And the security of AS can ensure that $CM^{i,z}_{\mathsf{ID}_j}$ is uniquely and securely stored. This fact can preliminarily convince the verifier at step 4 that the corresponding password has not been used before and will not be used at other places. The verifier can further verify the commitment at the step 6 once the password is securely opened within the channel $\mathsf{SCH}(\mathsf{ID}_j, \mathsf{V}_a)$. Last but not least, the identifier I_{V_a} can not only help the group member to the password-usage target (at the AS) but also hide the identity of the verifier to achieve the usage privacy of the password. So no one (incl. RA) can infer where the password is used from the values stored at the AS. Here we mainly present the proof idea of the above scheme. The formal security definition and the full proof are left as future work.

Other Security Considerations. For efficiency, we do not consider a separate open authority to handle the traceability of malicious group members in DGTOne. In some real-world settings, RA might not be fully trusted. A well-known way widely used in group digital schemes is to use two independent authorities to deal with the tasks of group member management and malicious member trace separately. This needs to incorporate the other authority known as *open authority* (OA). Like in many group digital signature schemes (e.g., [4,2]), a group member can use a public encryption scheme (PKE) to encrypt its identity based on OA's public key, and creates a proof using non-interactive zero knowledge (NIZK) system to show that the ciphertext is well-formed while carry out the join procedure with RA. In this way, it is possible to achieve public verifiable traceability as well. However, a side effect of such a solution is that it will introduce much more either storage or computation overheads (with caching or computing the PKE ciphertexts and NIZK proofs). So it may violate the original motivation of DGTOTP on providing lightweight anonymous authentication. Efficiency is our primary concern in this work. We leave the concrete DGTOTP construction with stronger security for future work.