Efficient Dynamic k-Times Anonymous Authentication

Lan Nguyen

CSIRO ICT Centre, Australia
WinMagic, Canada
Lan.Nguyen@winmagic.com

Abstract. In *k-times anonymous authentication* (k-TAA) schemes, members of a group can be anonymously authenticated to access applications for a bounded number of times determined by application providers. *Dynamic* k-TAA allows application providers to independently grant or revoke group members from accessing their applications. Dynamic k-TAA can be applied in several scenarios, such as k-show anonymous credentials, digital rights management, anonymous trial of Internet services, e-voting, e-coupons etc. This paper proposes the first provably secure dynamic k-TAA scheme, where authentication costs do not depend on k. This efficiency is achieved by using a technique called "efficient provable e-tag", proposed in [11], which could be applicable to other e-tag systems.
Keywords: privacy, anonymity, dynamic k-times anonymous authentication, k-show anonymous credentials, e-tag.

1 Introduction

In a k-times anonymous authentication system [13], participants include a group manager (GM), some application providers (AP) and many users. The GM registers users into the group and each AP independently announces the number of times a group member can access her application. A group member can be anonymously authenticated by APs within their allowed numbers of times and without contacting the GM. No one, even the GM or APs, is able to identify honest users or link two authentication executions of the same user while anyone can trace dishonest users. No party, even the GM, can successfully impersonate an honest user in an authentication execution.

However, k-TAA schemes are inflexible in the sense that the GM decides on the group membership and APs do not have any control over giving users access permission to their services. APs are passive and their role is limited to announcing the number of times a user can access their applications. In practice, APs want to select their user groups and grant or revoke access to users independently. For example, the AP may prefer to give access to users with good profile, or the AP may need to put an expiry date on users' access. *Dynamic* k-TAA [12] was introduced to provide these properties. In dynamic k-TAA, APs have more control over granting and revoking access to their services, and less

trust and computation from the GM is required. Dynamic k-TAA allows APs to restrict access to their services based on not only the number of times but also other factors such as expiry date and so can be used in much wider range of realistic scenarios.

A primitive close to k-TAA is Privacy-Protecting Coupon (PPC) system [6, 11], which consists of an *Initialisation* algorithm and 2 protocols, *Issue* and *Redeem*. There is a *vendor* and many *users*. The vendor can *issue* a k-redeemable coupon to a user such that the user can unlinkably *redeem* the coupon for exactly k times. There could be another algorithm, *Terminate*, which allows the vendor to *terminate* coupons. Compared to k-TAA, PPC does not allow traceability of malicious users and the vendor acts as the group manager and a single application provider.

Applications of k-TAA can be found in digital rights management (DRM). For example, k-TAA can be used to provide pay-per-use anonymous access to online digital content, such as music, movies, interactive games, betting and gambling, that are supplied by different application providers. A user can buy credits to download hundreds of songs or movies over a year at a discount price. Another example is trial browsing [13], where each provider allows members of a group, such as XXX community, to anonymously and freely browse content such as movies or music on trial. The provider also wants to limit the number of times that a user can access the service on trial and users, who try to go over the prescribed quota, must be identified.

The non-interactive counterpart of k-TAA is *k-times anonymous signatures* (i.e. *tracing-by-linking signatures* [15]), where a group member can anonymously sign messages on behalf of the group for k times. k-times anonymous signatures can be used to construct *k-show anonymous credential* systems [15], where credential-issuing organizations can limit the number of times a user can show her credentials. k-times anonymous signature can be applied to e-voting with limitation on the number of votes per user. It can be used to transfer e-cash: the cash owner sends an one-time anonymous signature on the cash to the receiver. It can also be directly used to construct an e-coupon scheme [13].

In previous k-TAA schemes, the authentication procedure has computation and communication costs linearly depending on the bound k. If an application provider sets k to be a large number, the authentication procedure becomes expensive. For example, a music web site may sell e-vouchers each of which can be used to anonymously download 10000 songs within a year. Then each user has to run the same expensive authentication protocol for each downloaded song. If there are many users in the group, the authentication cost multiplies by the number of users. So, the open problem is to construct k-TAA schemes where the computation and communication costs in the authentication procedure do not depend on k.

1.1 Our contribution

We propose the first dynamic k-TAA scheme with constant authentication costs, extended from the NS05 scheme [12], and prove its security. It can be used to

construct the first k-show anonymous credential system with constant costs. It can be converted to a k-TAA scheme using the approach in [12]. It is also possible to construct a combined scheme, where some of the APs have the dynamic property and other APs do not. Section 4.3 details efficiency comparison with previous k-TAA schemes [13, 12].

Our scheme still uses tag as in the TFS04 [13] and NS05 [12] schemes. In these schemes, the GM issues some secret key to each user. An AP with bound k provides a set of k tag bases. For each authentication, the user uses his secret key and a tag base to computes a value, called a tag, and sends it to the verifier with a zero-knowledge proof that the tag is correctly computed and the user is a group member. If the user attempts to access more than k times, he has to use a tag base twice and his identity will be revealed. The problem with these constructions is that the proof that the tag is correctly computed from one of the k tag bases requires a proof of knowledge of one of k elements and its cost linearly depends on k. Our objective is to remove this dependency.

We use a methodology, called "efficient provable e-tag", which was first proposed in [11] for a PPC system. An ordinary k-TAA scheme with constant costs [14] also uses this method.

In this method, each AP with bound k uses its secret key to issue k signatures on k random messages and these message-signature tuples are used as tag bases. Then the proof of knowledge of one of k elements is replaced by a proof of knowledge of a message-signature tuple. However, using our message-signature tuples with the function to compute tags from tag bases as in [13, 12] will result in a "cut and choose" zero-knowledge proof. So we use another function similar to the verifiable random function proposed in [7] that is used for the efficient compact e-cash scheme in [5]. We also need a different way for the GM to issue member secret and public keys to users.

The organization of the paper is as follows. We give the background in section 2 and present the model of dynamic k-TAA in section 3. Section 4 provides technical description of the proposed dynamic k-TAA scheme.

2 Preliminaries

We follow notation in [12, 13] and use some complexity assumptions, including Strong Diffie-Hellman (SDH), Decisional Bilinear Diffie-Hellman Inversion (DBDHI) and Computational Bilinear Diffie-Hellman Inversion 2 (CBDHI2). The notation and assumptions are provided in Appendix A.

2.1 Bilinear groups

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p. Suppose P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and there is an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\psi(P_2) = P_1$. A function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is said to be a bilinear pairing if it satisfies the following properties:

1. **Bilinearity:** $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.

2. **Non-degeneracy:** $e(P_1, P_2) \neq 1$.
3. **Computability:** $e(P, Q)$ is efficiently computed, $\forall P \in \mathbb{G}_1, Q \in \mathbb{G}_2$.

For simplicity, hereafter, we set $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ and $P_1 = P_2$ but the proposed scheme can be easily modified for $\mathbb{G}_1 \neq \mathbb{G}_2$. We define a Bilinear Pairing Instance Generator as a PPT algorithm $\mathcal{G}$ that takes 1^κ and returns a random tuple $\mathbf{t} = (p, \mathbb{G}, \mathbb{G}_T, e, P)$ of bilinear pairing parameters where p is of size κ.

2.2 General BB Signatures

This is a generalization of the Boneh-Boyen signature scheme [1], which is unforgeable under a weak chosen message attack if the SDH assumption holds. It allows generation of a single signature for two random messages and an efficient knowledge proof of the signature and messages without revealing anything about the signature and messages.

Key Generating. Suppose $(p, \mathbb{G}, \mathbb{G}_T, e, Q)$ is a bilinear pairing tuple. Generate random $H' \leftarrow \mathbb{G}$ and $s' \leftarrow \mathbb{Z}_p^*$ and obtain $Q'_{pub} = Q^{s'}$. The public key is (Q, H', Q'_{pub}) and the secret key is s'.

Signing. For messages $t \in \mathbb{Z}_p^*$ and $\check{t} \in \mathbb{Z}_p \setminus \{-s'\}$, output the signature $R = (Q^t H')^{1/(s'+\check{t})}$.

Verifying. For a public key (Q, H', Q'_{pub}), messages $t \in \mathbb{Z}_p^*$ and $\check{t} \in \mathbb{Z}_p \setminus \{-s'\}$, and a signature $R \in \mathbb{G}$, verify that $e(R, Q^{\check{t}} Q'_{pub}) = e(Q^t H', Q)$.

2.3 CL-SDH Signatures

This is a variant of the Camenisch-Lysyanskaya signature scheme [4] using the SDH assumption. Note that, as shown in [11], there is an efficient protocol between a user and a signer to generate a CL-SDH signature for the user's message without the signer learning anything about the message; and there is an efficient zero-knowledge proof of knowledge of a CL-SDH message-signature pair.

Key Generating. Suppose $(p, \mathbb{G}, \mathbb{G}_T, e, P)$ is a bilinear pairing tuple. Generate random $P_0, H' \leftarrow \mathbb{G}$ and $\gamma \leftarrow \mathbb{Z}_p^*$ and obtain $P_{pub} = P^\gamma$. The public key is (P, P_0, H', P_{pub}) and the secret key is γ.

Signing. For message $x \in \mathbb{Z}_p^*$, generate random $v \leftarrow \mathbb{Z}_p$ and $a \leftarrow \mathbb{Z}_p \setminus \{-\gamma\}$ and compute $S = (P^x H'^v P_0)^{1/(\gamma+a)}$. The signature is (a, S, v).

Verifying. For a public key (P, P_0, H', P_{pub}), a message $x \in \mathbb{Z}_p^*$, and a signature (a, S, v), verify that $e(S, P^a P_{pub}) = e(P^x H'^v P_0, P)$.

3 Model

This section revises the formal model for dynamic k-TAA [13, 12].

3.1 Procedures

A dynamic k-TAA system is specified as a tuple of PT algorithms (GKg, AKg, Join^U, Join^M, Bound, Grant, Revoke, Authen^U, Authen^P, Trace), operated by a *group manager* (GM), *application providers* (AP) and *users*. Each AP $\mathcal{V}$ has a public *authentication log* $\mathsf{LOG}_\mathcal{V}$, an *access group* $AG_\mathcal{V}$ of users who are allowed to access its application, and some *public information* $PI_\mathcal{V}$. The algorithms are described as follows.

GKg: The GM runs this *setup* PPT algorithm on input 1^l to obtain a group public key gpk and the GM's secret key gsk.

AKg: An AP $\mathcal{V}$ runs this PPT algorithm on input a group public key gpk to obtain a pair of AP public key and secret key $(apk_\mathcal{V}, ask_\mathcal{V})$.

Join^U, Join^M: This *joining* protocol allows the GM to register a user into the group. Both of the interactive algorithms Join^U (the user) and Join^M (the GM) take as input the group public key gpk and Join^M is also given the GM's secret key gsk. Join^M returns either accept or reject. If it is accept, Join^U outputs a pair of member public key and secret key (mpk_i, msk_i).

Bound: An AP $\mathcal{V}$ uses this *bound announcement* PPT algorithm to announce the number of times a user in its access group can use its application. It takes as input gpk, $apk_\mathcal{V}$ and $ask_\mathcal{V}$ and outputs the upper bound k and some information which is published with the AP's identity $ID_\mathcal{V}$.

Grant: An AP $\mathcal{V}$ runs this algorithm to grant a group member access to its application. The AP adds the member to its access group $AG_\mathcal{V}$ and updates his public information $PI_\mathcal{V}$. From $PI_\mathcal{V}$, the member can obtain an access key mak.

Revoke: This algorithm allows an AP to revoke a group member from accessing its application. It removes the member from the AP's access group and updates its public information.

Authen^U, Authen^P: This *authentication* protocol, between a user (Authen^U) and an AP $\mathcal{V}$ (Authen^P), allows the AP to authenticate the user for accessing its application. The protocol input is all of the AP and group's public information, and Authen^U's private input includes the user's keys mpk, msk and mak. Authen^P returns accept, if the user is in the AP's access group and has been authenticated by the AP less than k times, or reject otherwise. The authentication transcript is added to the log $\mathsf{LOG}_\mathcal{V}$.

Trace: Anyone can run this *public tracing* PPT algorithm to trace a malicious user. It takes as input all group public information and an authentication log and outputs a user identity, GM or NONE, which respectively mean "the user attempts to access more than the announced bound", "the GM published information maliciously", and "there is no malicious entity recorded in this log".

3.2 Correctness and Security Requirements

Brief descriptions of oracles $\mathcal{O}_{LIST}$, $\mathcal{O}_{QUERY}$, $\mathcal{O}_{JOIN-U}$, $\mathcal{O}_{AUTH-U}$, $\mathcal{O}_{JOIN-GM}$, $\mathcal{O}_{AUTH-AP}$, $\mathcal{O}_{GRAN-AP}$, $\mathcal{O}_{REVO-AP}$ and $\mathcal{O}_{CORR-AP}$ are provided in Ap-

pendix B. The correctness condition and security requirements for dynamic k-TAA are summarized as follows and full description can be found in [13, 12].

Correctness: It requires that an honest member who is in the access group of an honest AP and has performed the authentication protocol with the AP for less than the allowed number of times, is successfully authenticated by the AP.

Anonymity: Intuitively, it means that given two honest group members i_0 and i_1, who are in the access group of an AP, it is computationally hard to distinguish between authentication executions, which are performed by the AP and one of the two members. In the experiment, the adversary is allowed to collude with the GM, all APs, and all users except target users i_0 and i_1, and to query oracles $\mathcal{O}_{LIST}$, $\mathcal{O}_{JOIN-U}$, $\mathcal{O}_{AUTH-U}$ and $\mathcal{O}_{QUERY}$. The adversary is allowed to make only one query to $\mathcal{O}_{QUERY}$ on input i_0, i_1 and an AP whose access group contains i_0 and i_1. On receiving such a query, $\mathcal{O}_{QUERY}$ makes either i_0 or i_1 to execute the authentication protocol with the AP and outputs the protocol transcript. Each of the users i_0 and i_1 must be authenticated by the AP within k times. The anonymity condition holds if the probability that the adversary can correctly guess the user identity used in $\mathcal{O}_{QUERY}$'s authentication execution is negligibly better than a random guess.

This anonymity definition is general enough to capture desirable privacy properties. For example, if the adversary can link authentication executions of the same user with different APs with non-negligible probability, then the adversary can break the anonymity experiment with non-negligible probability. In the experiment, the adversary can use $\mathcal{O}_{AUTH-U}$ to trigger authentication executions between i_0 or i_1 with different APs. When $\mathcal{O}_{QUERY}$ generates a challenged authentication execution, the adversary can link it to the executions generated by $\mathcal{O}_{AUTH-U}$ with non-negligible probability. As the adversary knows the user identity of each execution generated by $\mathcal{O}_{AUTH-U}$, it can tell the user identity of the challenged authentication execution with non-negligible probability.

Detectability: It loosely means that if a subgroup of corrupted members have performed the authentication procedure with the same honest AP for more than the total allowed number of times, then the public tracing algorithm using the AP's authentication log outputs NONE with negligible probability. The experiment has two stages and the adversary is allowed to corrupt all users. In the first stage, the adversary can query $\mathcal{O}_{LIST}$, $\mathcal{O}_{JOIN-GM}$, $\mathcal{O}_{AUTH-AP}$, $\mathcal{O}_{GRAN-AP}$, $\mathcal{O}_{REVO-AP}$ and $\mathcal{O}_{CORR-AP}$. Then all authentication logs of all APs are emptied. In the second stage, the adversary continues the experiment, but without access to the revoking oracle $\mathcal{O}_{REVO-AP}$. The adversary wins if he can be successfully authenticated by an honest AP $\mathcal{V}$ with access bound k for more than $k \times \#AG_{\mathcal{V}}$ times, where $\#AG_{\mathcal{V}}$ is the number of members in the AP's access group. The detectability condition requires that the probability that the adversary wins is negligible.

Exculpability for users: It intuitively means that the tracing algorithm does not output the identity of an honest user even if other users, the GM and all APs are corrupted. In the experiment, the adversary, who wants to frame an honest user i, is allowed to corrupt all entities except the user i and can access

$\mathcal{O}_{LIST}$, $\mathcal{O}_{JOIN-U}$, and $\mathcal{O}_{AUTH-U}$. The adversary must authenticate user i using $\mathcal{O}_{AUTH-U}$ within the allowable numbers of times set by the APs. If the adversary succeeds in computing an authentication log, with which the public tracing algorithm outputs i, the adversary wins. The exculpability condition for users requires that the probability that the adversary wins is negligible.

Exculpability for the GM: Loosely speaking, it means that the tracing algorithm does not output the honest GM even if all users and all APs are corrupted. In the experiment, the adversary wants to frame the honest GM and he is allowed to corrupt all users and all APs and access $\mathcal{O}_{LIST}$ and $\mathcal{O}_{JOIN-GM}$. If the adversary succeeds in computing an authentication log, with which the public tracing algorithm outputs GM, the adversary wins. The exculpability condition for the GM requires that the probability that the adversary wins is negligible.

4 Dynamic k-TAA with constant authentication costs

4.1 Overview

Section 1.1 has already given the general intuition of the approach "efficient provable e-tag", which substantially improves efficiency of our scheme over the NS05 and TFS04 schemes [12, 13]. We now provide an outline of this scheme and note where this scheme is similar to NS05. In the GKg algorithm, a bilinear pairing tuple $(p, \mathbb{G}, \mathbb{G}_T, e, P)$ is generated, the GM's secret key is a CL-SDH secret key $\gamma \leftarrow \mathbb{Z}_p^*$ and the group public key includes the corresponding CL-SDH public key (P, P_0, H', P_{pub}) and a value $\Phi \leftarrow \mathbb{G}_T$.

As noted in section 2.3, there is an efficient protocol between a user and a signer to generate a CL-SDH signature for the user's secret message x without the signer learning anything about the message. This protocol underlies the joining protocol (Join^U, Join^M), where the user also has to publish his identity and $\beta = \Phi^{1/x}$ in the identification list LIST that allows tracing of malicious users in the Trace algorithm. At the end of the joining protocol, the user obtains a CL-SDH signature (a, S, v) for a message x, where v is also the user's random secret. The user's member secret key is (x, v) and his member public key is (a, S, β). As also noted in section 2.3, there is an efficient zero-knowledge proof of knowledge of a CL-SDH message-signature pair (by proving the knowledge of (a, S, v) and x such that $e(S, P^a P_{pub}) = e(P^x H'^v P_0, P)$). The user can be anonymously authenticated as a group member by using this proof, as shown in the authentication protocol.

In the AKg algorithm, an AP's public-secret key pair includes a general BB public key (Q, H', Q'_{pub}) and the corresponding BB secret key is s'. The Bound algorithm, for a bound k, generates k random message couples $(t_1, \check{t}_1), ..., (t_k, \check{t}_k)$ and k corresponding general BB signatures $R_1, ..., R_k$. The AP publishes k *tag bases* $(t_1, \check{t}_1, R_1), ..., (t_k, \check{t}_k, R_k)$ to be used for up to k times user access to the AP's service (each tag base is a general BB message-signature triplet).

In the authentication protocol between the AP and a group member with key pair $((x, v), (a, S, \beta))$, the user obtains a random l from the AP, chooses a

tag base $(t_i, \check{t}_i, R_i)$ and sends back a tag $(\Gamma, \check{\Gamma}) = (F(x, t_i), \check{F}(x, \check{t}_i, l))$, where F and $\check{F}$ are two functions. The user also shows the AP a zero-knowledge proof $Proof_2$ which proves four properties: (i) the user is a group member (by proving knowledge of a CL-SDH message-signature pair $(x, (a, S, v))$); (ii) the user knows a general BB message-signature triplet $(t_i, \check{t}_i, R_i)$ (without revealing the triplet); (iii) $(\Gamma, \check{\Gamma})$ is correctly computed from l, x, $(t_i, \check{t}_i)$, F and $\check{F}$ (that means $(\Gamma, \check{\Gamma}) = (F(x, t_i), \check{F}(x, \check{t}_i, l))$); and (iv) the AP has granted access to the user. Part (iv) is the same as in NS05 and we will talk about it afterwards. This protocol differs from NS05's authentication protocol with the construction of F and $\check{F}$ and parts (i), (ii) and (iii).

In the authentication protocols of TFS04 and NS05, the proof that one of the k announced tag bases has been used to compute the tag requires a proof of knowledge of one of k elements and its cost linearly depends on k. In our authentication protocol, that proof of knowledge of one of k tag bases is replaced by the proof of knowledge of a general BB message-signature triplet. Therefore, our authentication cost does not depend on k. The general BB signatures prevents the user from forging a new tag base without colluding the AP.

Similar to NS05, if the user uses the same tag base to compute another tag $(\Gamma', \check{\Gamma}')$, anyone can find these from the AP's authentication log (since $\Gamma = \Gamma'$) and use it to compute $\beta = (\check{\Gamma}/\check{\Gamma}')^{1/(l-l')}$, which is part of the user's public key ($\check{F}$ must be designed to allow this computation). However, if the member does not use the same tag base twice, his anonymity is protected (F and $\check{F}$ must be designed to allow this anonymity). The cost of checking if Γ has already appeared in the AP's authentication log is the same as in TFS04 and NS05, and is trivial if tags are orderly indexed by Γ, so we ignore that cost in claiming the 'constant' property.

F and $\check{F}$ must be designed so that: tags are not linkable; the property (iii) can be efficiently proved; and if a user uses the same tag base twice, his public key is computable from the two tags ($\beta = (\check{\Gamma}/\check{\Gamma}')^{1/(l-l')}$). We construct these two functions as $(\Gamma, \check{\Gamma}) = (\Phi^{1/(x+t_i)}, \Phi^{(lx+l\check{t}_i+x)/(x^2+x\check{t}_i)})$. This tag construction is different from [13, 12] and developed from a recently proposed verifiable random function [7] using bilinear pairings. It possesses a precious feature of having both key x and tag base $t_i, \check{t}_i$ in the exponents of $\Phi^{1/(x+t_i)}$ and $\Phi^{(lx+l\check{t}_i+x)/(x^2+x\check{t}_i)}$. This feature allows the user's zero-knowledge proof $Proof_2$ in the authentication protocol to avoid the cut-and-choose method.

Now, we talk about the property (iv) and the Grant and Revoke algorithms, which are quite the same as in NS05. We also use dynamic accumulators to provide the dynamic property, which means the AP grants access to or revokes access from users. Each AP has a public key/secret key pair $((Q, Q_{pub}), s)$, where $Q_{pub} = Q^s$. To grant access to a member with a public key (a, S, β), the AP accumulates the value a of the public key into an *accumulated value* $V \leftarrow V^{s+a}$, and the member obtains the old accumulated value as the witness W. The member shows that the AP has granted access to him by proving the knowledge of (a, W) such that $e(W, Q^a Q_{pub}) = e(V, Q)$. To revoke access from the member, the AP computes a new *accumulated value* $V \leftarrow V^{1/(s+a)}$.

Similar to NS05, there is a Public Inspection algorithm (Appendix C) executable by anyone to check if the APs perform the Bound, Grant and Revoke algorithms correctly.

4.2 Description

GKg.
On input 1^κ, the Bilinear Pairing Instance Generator returns $(p, \mathbb{G}, \mathbb{G}_T, e, P)$. Generate $P_0, P_1, P_2, H, H' \leftarrow \mathbb{G}$, $\gamma \leftarrow \mathbb{Z}_p^*$ and $\Phi \leftarrow \mathbb{G}_T$, and let $P_{pub} = P^\gamma$. The GM's secret key is a CL-SDH secret key $gsk = \gamma$. The group public key gpk consists of the corresponding CL-SDH group public key (P, P_0, H', P_{pub}) and values Φ, H, P_1, P_2. The identification list LIST of group members is initially empty.

AKg.
An AP $\mathcal{V}$ generates $Q \leftarrow \mathbb{G}$, $s, s' \leftarrow \mathbb{Z}_p^*$ and computes $Q_{pub} = Q^s$, $Q'_{pub} = Q^{s'}$. The public and secret keys for the AP are $apk = (Q, Q_{pub}, Q'_{pub})$ and $ask = (s, s')$, respectively. They form a general BB key pair $((Q, H', Q'_{pub}), s')$. Then, same as NS05 [12], AP maintains an authentication log LOG, an *accumulated value*, which is published and updated after granting or revoking a member, and a public archive ARC (as the other public information PI in the formal model), which is a list of 3-tuples. The first component of the tuple is an element in the public key of a member, who was granted or revoked from accessing the AP. The second component is a single bit indicating whether the member was granted (1) or revoked (0). The third component is the accumulated value after granting or revoking the member. Initially, the accumulated value is set to $V_0 \leftarrow \mathbb{G}$ and LOG and ARC are empty.

Join^U, Join^M.
This protocol allows the GM to generate a CL-SDH signature (a, S, v) for the user's secret x without learning anything about (x, v). The user also publishes $\beta = \Phi^{1/x}$. A user U_i can join the group as follows.

1. User U_i chooses $x, v' \leftarrow \mathbb{Z}_p^*$, computes $\beta = \Phi^{1/x}$ and a commitment $C = P^x H'^{v'}$ of x and adds (i, β) to the identification list LIST. The user then sends β and C to the GM with a standard non-interactive zero-knowledge proof $Proof_1 = PK\{(x, v') : C = P^x H'^{v'} \wedge \Phi = \beta^x\}$.
2. The GM verifies that (i, β) is an element of LIST and the proof is valid. The GM then generates $a \leftarrow \mathbb{Z}_p$ different from all corresponding previously generated values and $\tilde{v} \leftarrow \mathbb{Z}_p^*$, computes $S = (CH'^{\tilde{v}} P_0)^{1/(\gamma+a)}$, and sends $(S, a, \tilde{v})$ to user U_i.
3. User U_i computes $v = v' + \tilde{v}$ and confirms that equation $e(S, P^a P_{pub}) = e(P^x H'^v P_0, P)$ is satisfied. The new member U_i's secret key is $msk = (x, v)$, and his public key is $mpk = (a, S, \beta)$.

Bound.
An AP publishes his identity ID and a number k as the bound. Let $(t_j, \check{t}_j) =$

$\mathcal{H}_{\mathbb{Z}_p^* \times \mathbb{Z}_p^*}(ID, k, j)$ for $j = 1, ..., k$. The AP computes general BB signatures $R_j = (Q^{t_j} H')^{1/(s'+\check{t}_j)}$ for $j = 1, ..., k$ and publishes $(t_1, \check{t}_1, R_1), ..., (t_k, \check{t}_k, R_k)$. We call $(t_j, \check{t}_j, R_j)$ the j^{th} *tag base* of the AP.

The Public Inspection algorithm (Appendix C) can be run by anyone to check if the APs perform the Bound, Grant and Revoke algorithms correctly. So it is negligible that the APs can generate tag bases maliciously, for example, two APs setting the same t_j.

Grant.
This is the same as in NS05. An AP grants access to a user U_i with public key $mpk = (a, \cdot, \cdot)$ as follows. Suppose there are j tuples in the AP's ARC and the AP's current accumulated value is V_j. The AP computes a new accumulated value $V_{j+1} = V_j^{s+a}$ and adds $(a, 1, V_{j+1})$ to his ARC. From the AP's ARC, the user U_i forms his access key $mak = (j + 1, W)$, where $W = V_j$, and keeps a counter d, which is initially set to 0.

Revoke.
This is the same as in NS05. An AP revokes access from a user U_i with public key $mpk = (a, \cdot, \cdot)$ as follows. Suppose there are j tuples in the AP's ARC and the AP's current accumulated value is V_j. The AP computes a new accumulated value $V_{j+1} = V_j^{1/(s+a)}$, and adds $(a, 0, V_{j+1})$ to ARC.

Authen^U, Authen^P.
The difference from NS05's authentication protocol lies in the second step, which is also the most important step of the protocol. In this step, the tag computation and $Proof_2$ are completely different from those in NS05. An AP (ID, k), whose public key and current accumulated value are $apk = (Q, Q_{pub}, Q'_{pub})$ and V respectively, authenticates a user U with public and secret keys $mpk = (a, S, \beta)$ and $msk = (x, v)$, respectively, as follows.

1. U increases counter d. If $d > k$, then U sends $\perp$ to the AP and stops. Otherwise, U runs the algorithm Update (Appendix C) to update his access key $mak = (j, W)$. The AP then sends a random integer $l \leftarrow \mathbb{Z}_p^*$ to U.
2. U chooses an unused tag base $(t_\iota, \check{t}_\iota, R_\iota)$, computes tag $(\Gamma, \check{\Gamma}) = (\Phi^{1/(x+t_\iota)}, \Phi^{(lx+l\check{t}_\iota+x)/(x^2+x\check{t}_\iota)})$, and sends $(\Gamma, \check{\Gamma})$ to the AP with a proof $Proof_2 = PK\{(t_\iota, \check{t}_\iota, R_\iota, a, S, x, v, W) : \Gamma = \Phi^{1/(x+t_\iota)} \wedge \check{\Gamma} = \Phi^{(lx+l\check{t}_\iota+x)/(x^2+x\check{t}_\iota)} \wedge e(S, P^a P_{pub}) = e(P^x H'^v P_0, P) \wedge e(W, Q^a Q_{pub}) = e(V, Q) \wedge e(R_\iota, Q^{\check{t}_\iota} Q'_{pub}) = e(Q^{t_\iota} H', Q)\}$ ($Proof_2$ is described below).
3. If the proof is valid and if Γ is different from all corresponding tags in the AP's LOG, the AP adds tuple $(\Gamma, \check{\Gamma}, l)$ and the proof to LOG, and outputs accept. If the proof is valid and Γ is already written in LOG, the AP adds tuple $(\Gamma, \check{\Gamma}, l)$ and the proof to the LOG, outputs (detect,LOG) and stops. If the proof is invalid, the AP outputs reject and stops.

Proof$_2$.
Let $U_1 = SH^{r_1}$; $U_2 = WH^{r_2}$; $U_3 = R_\iota H^{r_3}$ where $r_1, r_2, r_3 \leftarrow \mathbb{Z}_p$, then $Proof_2$

is equivalent to a proof of knowledge of $(t_\iota, \check{t}_\iota, a, x, v, r_1, r_2, r_3)$ such that

$$\Gamma^{x+t_\iota} = \Phi;\ \check{\Gamma}^{(x+\check{t}_\iota)x}\Phi^{-lx-l\check{t}_\iota-x} = 1;$$
$$e(U_1,P)^a e(H,P)^{-r_1 a} e(H,P_{pub})^{-r_1} e(P,P)^{-x} e(H',P)^{-v}$$
$$= e(U_1,P_{pub})^{-1} e(P_0,P);$$
$$e(U_2,Q)^a e(H,Q)^{-r_2 a} e(H,Q_{pub})^{-r_2} = e(U_2,Q_{pub})^{-1} e(V,Q);$$
$$e(U_3,Q)^{\check{t}_\iota} e(H,Q)^{-r_3\check{t}_\iota} e(H,Q'_{pub})^{-r_3} e(Q,Q)^{-t_\iota} = e(U_3,Q'_{pub})^{-1} e(H',Q)$$

Most of the pairing operations in this proof can be pre-computed. The member M computes the proof as follows.

1. Generate $r_1, r_2, r_3, k_1, ..., k_{18} \leftarrow \mathbb{Z}_p$ and compute
   $U_1 = SH^{r_1}$; $U_2 = WH^{r_2}$; $U_3 = R_\iota H^{r_3}$;
   $U_4 = P_1^{r_1} P_2^{r_2} H^{r_4}$; $U_5 = P_1^{r_3} H^{r_5}$; $U_6 = P_1^{x+\check{t}_\iota} H^{r_6}$;
   $T_1 = P_1^{k_1} P_2^{k_2} H^{k_4}$; $T_2 = P_1^{k_7} P_2^{k_8} H^{k_9} U_4^{-k_{10}}$; $T_3 = P_1^{k_3} H^{k_5}$;
   $T_4 = P_1^{k_{11}} H^{k_{12}} U_5^{-k_{13}}$; $T_5 = P_1^{k_{14}+k_{13}} H^{k_6}$; $T_6 = P_1^{k_{15}} H^{k_{16}} U_6^{-k_{14}}$;
   $\Pi_1 = \Gamma^{k_{14}+k_{17}}$; $\Pi_2 = \check{\Gamma}^{k_{15}}\Phi^{-lk_{14}-lk_{13}-k_{14}}$;
   $\Pi_3 = e(U_1,P)^{k_{10}} e(H,P)^{-k_7} e(H,P_{pub})^{-k_1} e(P,P)^{-k_{14}} e(H',P)^{-k_{18}}$;
   $\Pi_4 = e(U_2,Q)^{k_{10}} e(H,Q)^{-k_8} e(H,Q_{pub})^{-k_2}$;
   $\Pi_5 = e(U_3,Q)^{k_{13}} e(H,Q)^{-k_{11}} e(H,Q'_{pub})^{-k_3} e(Q,Q)^{-k_{17}}$
2. Compute $c = \mathcal{H}_{\mathbb{Z}_p}(P||P_{pub}||P_0||H||H'||P_1||P_2||\Phi||Q||Q_{pub}||Q'_{pub}||ID||k||l||$
   $V||U_1||...||U_6||T_1||...||T_6||\Pi_1||...||\Pi_5)$
3. Compute in $\mathbb{Z}_p$: $s_1 = k_1 + cr_1$; $s_2 = k_2 + cr_2$; $s_3 = k_3 + cr_3$; $s_4 = k_4 + cr_4$; $s_5 = k_5 + cr_5$; $s_6 = k_6 + cr_6$; $s_7 = k_7 + cr_1 a$; $s_8 = k_8 + cr_2 a$; $s_9 = k_9 + cr_4 a$; $s_{10} = k_{10} + ca$; $s_{11} = k_{11} + cr_3\check{t}_\iota$; $s_{12} = k_{12} + cr_5\check{t}_\iota$; $s_{13} = k_{13} + c\check{t}_\iota$; $s_{14} = k_{14} + cx$; $s_{15} = k_{15} + c(x+\check{t}_\iota)x$; $s_{16} = k_{16} + cr_6 x$; $s_{17} = k_{17} + ct_\iota$; $s_{18} = k_{18} + cv$
4. Output $(U_1, ..., U_6, c, s_1, ..., s_{18})$

Verification of $Proof_{2b}$. Checking the following equation

$$c \stackrel{?}{=} \mathcal{H}_{\mathbb{Z}_p}(P||P_{pub}||P_0||H||H'||P_1||P_2||\Phi||Q||Q_{pub}||Q'_{pub}||ID||k||l||V||U_1||...||U_6||$$
$$P_1^{s_1} P_2^{s_2} H^{s_4} U_4^{-c}||P_1^{s_7} P_2^{s_8} H^{s_9} U_4^{-s_{10}}||P_1^{s_3} H^{s_5} U_5^{-c}||P_1^{s_{11}} H^{s_{12}} U_5^{-s_{13}}||$$
$$P_1^{s_{14}+s_{13}} H^{s_6} U_6^{-c}||P_1^{s_{15}} H^{s_{16}} U_6^{-s_{14}}||\Gamma^{s_{14}+s_{17}}\Phi^{-c}||\check{\Gamma}^{s_{15}}\Phi^{-ls_{14}-ls_{13}-s_{14}}||$$
$$e(U_1,P)^{s_{10}} e(H,P)^{-s_7} e(H,P_{pub})^{-s_1} e(P,P)^{-s_{14}} e(H',P)^{-s_{18}} e(U_1,P_{pub})^c$$
$$e(P_0,P)^{-c}||e(U_2,Q)^{s_{10}} e(H,Q)^{-s_8} e(H,Q_{pub})^{-s_2} e(U_2,Q_{pub})^c e(V,Q)^{-c}||$$
$$e(U_3,Q)^{s_{13}} e(H,Q)^{-s_{11}} e(H,Q'_{pub})^{-s_3} e(Q,Q)^{-s_{17}} e(U_3,Q'_{pub})^c e(H',Q)^{-c}.$$

Trace.
This algorithm is almost the same as in NS05. The identity of a malicious user can be traced from an AP's LOG as follows.

1. Look for two entries $(\Gamma, \check{\Gamma}, l, Proof)$ and $(\Gamma', \check{\Gamma}', l', Proof')$ in the LOG, such that $\Gamma = \Gamma'$ and $l \neq l'$, and that $Proof$ and $Proof'$ are valid. If no such entry can be found, output NONE.
2. Compute $\beta = (\check{\Gamma}/\check{\Gamma}')^{1/(l-l')} =$
   $(\Phi^{(lx+l\check{t}_\iota+x)/(x^2+x\check{t}_\iota)}/\Phi^{(l'x+l'\check{t}'_\iota+x)/(x^2+x\check{t}'_\iota)})^{1/(l-l')} = \Phi^{1/x}$, and look for a pair (i,β) from the LIST. Output member identity i, or if no such (i,β) can be found conclude that the GM has deleted some data from LIST, and output GM.

4.3 Comparison

Apart from providing the same desirable properties of the NS05 and TFS04 schemes, a significant advantage of our scheme is that its authentication costs do not depend on k or any parameter. Its only tradeoff is that the Bound algorithm needs to compute $\{R_1, ..., R_k\}$ for the tag bases. However, each AP needs to run the Bound algorithm only once whereas the authentication protocol is executed by all granted members for k times. So the tradeoff is very trivial compared to the advantage.

We have the following comparison on the number of exponentiations (EX), scalar multiplications (SM), pairings (PA) and transmitted bytes in the authentication protocol. For the communication comparison, we use the parameters in [12]. The TFS04 scheme has $\nu = 1024$, $\varepsilon = \mu = \kappa = 160$. For other schemes, p is a 160-bit prime, $\mathbb{G}_T$ is a subgroup of a finite field of size approximately 2^{1024} and $\mathbb{G}_T$ elements can be compressed by a factor of three using techniques in [9]. Most of the pairings can be pre-computed. The user can compute $e(U_1, P)^{k_{10}}$ by pre-computing $e(S, P)$ and $e(H, P)$ and computing $e(S, P)^{k_{10}} e(H, P)^{k_{10} r_1}$ (this way removes pairing computation but increases the number of exponentiations). It is similar for $e(U_2, Q)^{k_{10}}$ and $e(U_3, Q)^{k_{13}}$. Note that the TFS04 scheme does not provide the dynamic property and does not have the Update algorithm. That algorithm is the same for NS05 and our scheme. It is not needed if NS05 and our scheme are modified to remove the dynamic property. So we do not count the cost of the Update algorithm in the comparison table. Besides, the number of bytes sent by a user in the NS05 scheme we computed (60 k+ 408) is different from that in [12] (60 k+ 304).

	TFS04	NS05	Our scheme
Computation by AP	(17+8k)EXs	(15+8k)EXs +8SMs+4PAs	21EXs+ 20SMs+6PAs
Computation by User	(28+8k)EXs	(21+8k)EXs +12SMs	22EXs+ 27SMs
Bytes sent by AP	40	20	20
Bytes sent by User	60 k+ 1617	60 k+ 408	585
Dynamic	No	Yes	Yes

4.4 Security

Security of our scheme is stated in Theorem 1, which is proved in Appendix D.

Theorem 1. *In the random oracle model, the dynamic k-TAA scheme provides: (i) Correctness; (ii) Anonymity under the Decisional Bilinear Diffie-Hellman Inversion assumption; (iii) Detectability under the Strong Diffie-Hellman assumption; (iv) Exculpability for users under the Computational Bilinear Diffie-Hellman Inversion 2 assumption; (v) Exculpability for the GM under the Strong Diffie-Hellman assumption.*

References

1. D. Boneh and X. Boyen. Short Signatures Without Random Oracles. EUROCRYPT 2004, Springer-Verlag, LNCS 3027, pp. 56-73.
2. D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. EUROCRYPT 2004, Springer-Verlag, LNCS 3027, pp. 223-238.
3. J. Camenisch, and A. Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. CRYPTO 2002, Springer-Verlag, LNCS 2442, pp. 61-76.
4. J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. SCN 2002, Springer-Verlag, LNCS 2576.
5. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. EUROCRYPT 2005, Springer-Verlag, LNCS 3494, pp. 302-321, 2005.
6. L. Chen, M. Enzmann, A. Sadeghi, M. Schneider, and M. Steiner. A Privacy-Protecting Coupon System. Financial Cryptography 2005, Springer-Verlag, LNCS 3570, pp. 93-109.
7. Y. Dodis and A. Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. Public Key Cryptography 2005, Springer-Verlag, LNCS 3386, pp. 416-431.
8. A. Fiat, and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. CRYPTO 1986, Springer-Verlag, LNCS 263, pp. 186-194.
9. R. Granger, D. Page, and M. Stam. A Comparison of CEILIDH and XTR. Algorithmic Number Theory, 6th International Symposium, ANTS-VI, pages 235-249. Springer, June 2004.
10. A. Kiayias, and Moti Yung. Group Signatures: Provable Security, Efficient Constructions and Anonymity from Trapdoor-Holders. Cryptology ePrint Archive: Report 2004/076.
11. Lan Nguyen. Privacy-Protecting Coupon System Revisited. Financial Cryptography Conference (FC) 2006, LNCS, Springer, 2006.
12. L. Nguyen and R. Safavi-Naini. Dynamic k-Times Anonymous Authentication. Applied Cryptography and Network Security (ACNS) 2005, Springer-Verlag, LNCS 3531, 2005.
13. I. Teranisi, J. Furukawa, and K. Sako. k-Times Anonymous Authentication. ASIACRYPT 2004, Springer-Verlag, LNCS 3329, pp. 308-322, 2004.
14. I. Teranishi and K. Sako. k-Times Anonymous Authentication with a Constant Proving Cost. Public Key Cryptography 2006, Springer-Verlag, LNCS 3958, pp. 525-542, 2006.
15. V. Wei. Tracing-by-Linking Group Signatures. Information Security Conference (ISC) 2005, Springer-Verlag, LNCS 3650, pp. 149-163, 2005.

A Preliminaries

A.1 Notation

For a function $f : \mathbb{N} \rightarrow \mathbb{R}^+$, if for every positive number α, there exists a positive integer κ_0 such that for every integer $\kappa > \kappa_0$, it holds that $f(\kappa) < \kappa^{-\alpha}$, then f is said to be *negligible*. Let PT denote *polynomial-time*, PPT denote *probabilistic* PT and DPT denote *deterministic* PT. For a PT algorithm $\mathcal{A}(\cdot)$, "$x \leftarrow A(\cdot)$" denotes an output from the algorithm. For a set $\mathbf{X}$, "$x \leftarrow \mathbf{X}$"

denotes an element uniformly chosen from $\mathbf{X}$, and $\#\mathbf{X}$ denotes the number of elements in $\mathbf{X}$. Let "$\mathsf{Pr}[Procedures|Predicate]$" denote the probability that $Predicate$ is true after executing the $Procedures$, $\mathcal{H}_{\mathbf{X}}$ denote a hash function from the set of all finite binary strings $\{0,1\}^*$ onto the set $\mathbf{X}$, and $PK\{x : R(x)\}$ denote a proof of knowledge of x that satisfies the relation $R(x)$. An *adversary* is modelled by an interactive Turing machine, which interacts with some oracles. Each oracle performs operations and produces outputs required by queries from the adversary. An entity is *corrupted* if the adversary has the entity's secret keys and completely controls the entity's actions. We define $1/0$ to be 0.

A.2 Complexity assumptions

q-Strong Diffie-Hellman (q-SDH) Assumption [1]. *For every PPT algorithm $\mathcal{A}$, the following function $Adv_{\mathcal{A}}^{q\text{-}SDH}(\kappa)$ is negligible.*

$$Adv_{\mathcal{A}}^{q\text{-}SDH}(\kappa) = Pr[(\mathcal{A}(\mathbf{t}, P, P^s, \ldots, P^{(s^q)}) = (c, P^{1/(s+c)})) \wedge (c \in \mathbb{Z}_p)]$$

where $\mathbf{t} = (p, \mathbb{G}, \mathbb{G}_T, e, P) \leftarrow \mathcal{G}(1^\kappa)$ *and* $s \leftarrow \mathbb{Z}_p^*$.

The assumption informally means that there is no PPT algorithm that can compute a pair $(c, P^{1/(s+c)})$, where $c \in \mathbb{Z}_p$, from a tuple $(P, P^s, \ldots, P^{(s^q)})$, where $s \leftarrow \mathbb{Z}_p^*$.

Decisional Bilinear Diffie-Hellman Inversion (DBDHI) Assumption [2]. *For every PPT algorithm $\mathcal{A}$, the following function $Adv_{\mathcal{A}}^{DBDHI}(\kappa)$ is negligible.*

$$\begin{aligned} Adv_{\mathcal{A}}^{DBDHI}(\kappa) = |&Pr[\mathcal{A}(\mathbf{t}, P, P^s, \ldots, P^{(s^q)}, e(P,P)^{1/s}) = 1] \\ &- Pr[\mathcal{A}(\mathbf{t}, P, P^s, \ldots, P^{(s^q)}, \Gamma) = 1]| \end{aligned}$$

where $\mathbf{t} = (p, \mathbb{G}, \mathbb{G}_T, e, P) \leftarrow \mathcal{G}(1^\kappa)$, $\Gamma \leftarrow \mathbb{G}_T^*$ *and* $s \leftarrow \mathbb{Z}_p^*$.

Intuitively the DBDHI assumption [2] states that there is no PPT algorithm that can distinguish between a tuple $(P, P^s, \ldots, P^{(s^q)}, e(P,P)^{1/s})$ and a tuple $(P, P^s, \ldots, P^{(s^q)}, \Gamma)$, where $\Gamma \leftarrow \mathbb{G}_T^*$ and $s \leftarrow \mathbb{Z}_p^*$. We define the Computational Bilinear Diffie-Hellman Inversion 2 assumption, which holds if either DBDHI or SDH holds.

Computational Bilinear Diffie-Hellman Inversion 2 (CBDHI2) Assumption. *For every PPT algorithm $\mathcal{A}$, the following function $Adv_{\mathcal{A}}^{CBDHI2}(\kappa)$ is negligible.*

$$Adv_{\mathcal{A}}^{CBDHI2}(\kappa) = Pr[\mathcal{A}(\mathbf{t}, P, P^s, \ldots, P^{(s^q)}, e(P,P)^{1/s}) = s]$$

where $\mathbf{t} = (p, \mathbb{G}, \mathbb{G}_T, e, P) \leftarrow \mathcal{G}(1^\kappa)$ *and* $s \leftarrow \mathbb{Z}_p^*$.

B Oracles

The adversary has access to a number of oracles and can query them according to the brief description below, to learn about the system and increase his success chance in the attacks. Their formal definitions can be found in [13, 12].

$\mathcal{O}_{LIST}$: Suppose there is an *identification list* LIST of user identity/public-key pairs, this oracle maintains correct correspondence between user identities and user public keys. Any party can query the oracle to view a user's public key. A user or his colluder can request the oracle to record the user's identity and public key to LIST. The GM or his colluder can request the oracle to delete data from LIST.

$\mathcal{O}_{QUERY}$: It is only used once in the definition for anonymity requirement to give the adversary a challenged authentication transcript. Identities of an AP and two honest users, who are in the AP's access group and have not been authenticated by the AP more than the limit, are given to the oracle. It then randomly chooses one of the two identities, executes the authentication protocol between the chosen identity and the AP, and outputs the transcript of the protocol.

Given a user identity, $\mathcal{O}_{JOIN-GM}$ performs the (Join^U, Join^M) protocol as executed by the honest GM and the user. Given an honest user's identity, $\mathcal{O}_{JOIN-U}$ performs the (Join^U, Join^M) protocol between the GM and the user. Given an honest AP's identity and a user identity, $\mathcal{O}_{AUTH-AP}$ makes the AP to execute the authentication protocol with the user. Given an honest user's identity and an AP identity, $\mathcal{O}_{AUTH-U}$ makes the user to perform the authentication protocol with the AP. $\mathcal{O}_{GRAN-AP}$ takes as input an honest AP's identity and a group member's identity and the AP executes the Grant algorithm to grant access to the user. $\mathcal{O}_{REVO-AP}$ takes as input an honest AP's identity and a member of the AP's access group and the AP executes the Revoke algorithm to revoke the user's access right. $\mathcal{O}_{CORR-AP}$ corrupts an AP specified in its input.

C Update and Public Inspection

Update.
This algorithm is the same as in NS05. Suppose the AP's ARC currently has n tuples, the member M with the public key $(a, \cdot, \cdot)$ and the access key (j, W_j) computes a new access key as follows.

for $(k = j + 1; k++; k \leq n)$ do
  retrieve from ARC the k^{th} tuple (u, b, V_k);
  if $b = 1$, then $W_k = V_{k-1} W_{k-1}^{u-a}$
  else $W_k = (W_{k-1}/V_k)^{1/(u-a)}$ end if;
end for;
return (n, W_n);

Public Inspection.
Not only checking Grant and Revoke as in NS05, this algorithm also checks the Bound algorithm. Any party can run this algorithm to assure the correctness of an AP's tag bases and public archive ARC. With such an algorithm, we can assume that tag bases are always correctly issued and ARC is always correctly updated. For each tag base $(t_j, \check{t}_j, R_j)$ of an AP ID with bound k, any party can

verify if $(t_j, \check{t}_j) \stackrel{?}{=} \mathcal{H}_{\mathbb{Z}_p^* \times \mathbb{Z}_p^*}(ID, k, j)$ and $e(R_j, Q^{\check{t}_j} Q'_{pub}) \stackrel{?}{=} e(Q^{t_j} H', Q)$. After a change on ARC, any party can retrieve the new tuple (u, b, V_k). If $(b = 1)$ then he checks if $e(V_{k-1}, Q^a Q_{pub}) \stackrel{?}{=} e(V_k, Q)$; otherwise, he checks if $e(V_k, Q^a Q_{pub}) \stackrel{?}{=} e(V_{k-1}, Q)$.

D Security Proofs

For Theorem 1, as part (i) can easily be proved by checking equations, we only provide proofs for parts (ii), (iii), (iv) and (v). Due to space limitation, we omit the proof that $Proof_2$ is non-interactive zero-knowledge, which is standard.

D.1 Proof of Theorem 1 (ii)

Suppose there exists a PPT adversary $\mathcal{A}$ breaking the Anonymity property of our scheme, we show a PPT adversary $\mathcal{B}$ that can break the DBDHI assumption. Let $\mathbf{t} = (p, \mathbb{G}, \mathbb{G}_T, e, P') \leftarrow \mathcal{G}(1^\kappa)$ and a tuple $\alpha = (P', P'^w, \ldots, P'^{(w^q)}, \Lambda)$ be uniformly chosen from either $S_0 = \{(P', P'^w, \ldots, P'^{(w^q)}, e(P', P')^{1/w}) | w \leftarrow \mathbb{Z}_p^*\}$ or $S_1 = \{(P', P'^w, \ldots, P'^{(w^q)}, \Lambda) | w \leftarrow \mathbb{Z}_p^*, \Lambda \leftarrow \mathbb{G}_T^*\}$. $\mathcal{B}$'s challenge is to guess whether α is chosen from S_0 or S_1. $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

$\mathcal{B}$ randomly chooses a bit $b \leftarrow \{0, 1\}$ and let b' be the other bit. $\mathcal{B}$ generates different $\delta_0, \check{\delta}_0, \delta_1, \check{\delta}_1, ..., \delta_{q-1}, \check{\delta}_{q-1} \leftarrow \mathbb{Z}_p^*$ and sets $x_b = w - \delta_0$ (without knowing x_b). Let $F = x_b(x_b + \check{\delta}_0) \prod_{i=1}^{q-1}(x_b + \delta_i)(x_b + \check{\delta}_i)$, then it can be presented as a polynomial $F = \sum_{i=0}^{2q} A_i w^i$, where $A_0, ..., A_{2q}$ are computable from $\delta_0, \check{\delta}_0, \delta_1, \check{\delta}_1, ..., \delta_{q-1}, \check{\delta}_{q-1}$ and $A_0 \neq 0$. Therefore, $\mathcal{B}$ can compute $\Phi = e(P', P')^F$, $\beta_b = \Phi^{1/x_b}$, $\check{\Theta}_0 = \Phi^{1/(x_b + \check{\delta}_0)}$, $\Theta_i = \Phi^{1/(x_b + \delta_i)}$ and $\check{\Theta}_i = \Phi^{1/(x_b + \check{\delta}_i)}$, $i = 1, ..., q-1$ from $(P', P'^w, \ldots, P'^{(w^q)})$. Given l, $\mathcal{B}$ can also compute $\Phi^{(lx_b + l\check{\delta}_i + x_b)/(x_b^2 + x_b\check{\delta}_i)} = \beta_b^l \check{\Theta}_i$ for $i = 0, ..., q-1$. Let $\Theta_0 = e(P', P')^{\sum_{i=1}^{2q} A_i w^{i-1}}$ Λ^{A_0}, if $\Lambda = e(P', P')^{1/w}$ then $\Theta_0 = \Phi^{1/(x_b + \delta_0)}$.

$\mathcal{B}$ selects $P, P_0, P_1, P_2, H, H' \leftarrow \mathbb{G}$, $\gamma \leftarrow \mathbb{Z}_p^*$, and computes $P_{pub} = P^\gamma$. $\mathcal{B}$ provides $\mathcal{A}$ the group public key $gpk = (P, P_0, H', P_{pub}, \Phi, H, P_1, P_2)$ and the group secret key $gsk = \gamma$. $\mathcal{B}$ creates a number of users including two target users i_0 and i_1 that will be sent to $\mathcal{O}_{QUERY}$ later.

At any time, $\mathcal{A}$ can create a new AP by generating apk, ask, an initial accumulated value, LOG and ARC as described in the AKg algorithm. Because $\mathcal{A}$ determines the AP's identity to be sent to $\mathcal{O}_{QUERY}$, it can create more APs without detriment to its attack. Therefore, let ζ be the upper bound on the number of APs, we can assume $\mathcal{A}$ always creates ζ APs. $\mathcal{B}$ randomly picks $m \leftarrow \mathbb{Z}_\zeta^*$. Suppose the m^{th} AP ID_m has bound k_m, $\mathcal{B}$ randomly picks $j_m \leftarrow \mathbb{Z}_{k_m}^*$.

$\mathcal{B}$ simulates oracles accessible by $\mathcal{A}$ as follows.

- Random oracle $\mathcal{H}_{\mathbb{Z}_p^* \times \mathbb{Z}_p^*}$: This oracle is queried in the Bound algorithm. If the query is (ID_m, k_m, j_m), the oracles returns $(t = \delta_0, \check{t} = \check{\delta}_0)$. Otherwise, on the i^{th} query, the oracle returns $(t = \delta_i, \check{t} = \check{\delta}_i)$.

- $\mathcal{O}_{LIST}$: This oracle operates as in the definition of $\mathcal{O}_{LIST}$, with regard to an *identification list* LIST of user identity/public-key pairs. $\mathcal{A}$ can query the oracle to view a user's public key. $\mathcal{A}$ can request the oracle to record the identity and public key of a user, who is not i_0 or i_1, to LIST. $\mathcal{A}$ can request the oracle to delete data from LIST.
- $\mathcal{O}_{JOIN-U}$: $\mathcal{A}$ just needs to query this oracle to register i_0 and i_1 to the group, as $\mathcal{A}$ can collude other users and the GM.
  If $\mathcal{A}$ asks the oracle to register i_b, $\mathcal{B}$ chooses $C_b \leftarrow \mathbb{G}$, computes $\beta_b = \Phi^{1/x_b}$ and adds (i_b, β_b) to LIST. $\mathcal{B}$ (the oracle) then returns β_b and C_b to $\mathcal{A}$ (the GM) with a simulation of the standard non-interactive zero-knowledge proof $Proof_1 = PK\{(x_b, v'_b) : C_b = P^{x_b} H'^{v'_b} \wedge \Phi = \beta_b^{x_b}\}$. The GM follows the Join protocol's description and sends back $(S_b, a_b, \tilde{v}_b)$. $\mathcal{B}$ then checks that $e(S_b, P^{a_b} P_{pub}) = e(C_b H'^{\tilde{v}_b} P_0, P)$ and sets i_b's public key as (a_b, S_b, β_b) (i_b's secret key $(x_b, v_b = v'_b + \tilde{v}_b)$ is unknown).
  If $\mathcal{A}$ asks the oracle to register $i_{b'}$, $\mathcal{B}$ chooses $x_{b'}, v'_{b'} \leftarrow \mathbb{Z}_p^*$, computes $\beta_{b'} = \Phi^{1/x_{b'}}$ and follows the Join protocol's description so that $(i_{b'}, \beta_{b'})$ is added to LIST, $i_{b'}$'s public key is $(a_{b'}, S_{b'}, \beta_{b'})$ and $i_{b'}$'s secret key is $(x_{b'}, v_{b'})$.
- $\mathcal{O}_{AUTH-U}$: $\mathcal{A}$ just needs to query this oracle to authenticate i_0 and i_1, as $\mathcal{A}$ can collude other users, the APs and the GM.
  If i_b is queried to be authenticated by an AP (ID, k), whose public key and current accumulated value are $apk = (Q, Q_{pub}, Q'_{pub})$ and V respectively, and i_b's counter d for this AP is not greater than k, $\mathcal{B}$ runs the algorithm Update to update his access key $mak = (j, W_b)$. On receiving a random integer $l \leftarrow \mathbb{Z}_p^*$ from the AP, $\mathcal{B}$ chooses a unused tag base $(t_\iota, \check{t}_\iota, R_\iota)$, where $(t_\iota, \check{t}_\iota)$ is different from $(\delta_0, \check{\delta}_0)$, computes tag $(\Gamma, \check{\Gamma}) = (\Phi^{1/(x_b + t_\iota)}, \Phi^{(l x_b + l\check{t}_\iota + x_b)/(x_b^2 + x_b \check{t}_\iota)})$, and sends $(\Gamma, \check{\Gamma})$ to the AP with a simulation of the proof $Proof_2$, which can be done by using the simulator in the proof for $Proof_2$'s zero-knowledge property and resetting the random oracle. $\mathcal{A}$ and $\mathcal{B}$ perform the rest of the authentication protocol as specified in Section 4.2.
  If $i_{b'}$ is queried to be authenticated by an AP, as $\mathcal{B}$ knows $i_{b'}$'s secret key, $\mathcal{A}$ and $\mathcal{B}$ can simulate the authentication protocol as specified in Section 4.2.
- $\mathcal{O}_{QUERY}$: If the queried AP is not ID_m, $\mathcal{B}$ fails and exits. Otherwise, as m is randomly chosen, the probability that the queried AP is ID_m is at least $1/\zeta$. In this case, suppose the AP ID_m has public key $apk = (Q, Q_{pub}, Q'_{pub})$ and current accumulated value V, and i_b's counter d for this AP is not greater than k. $\mathcal{B}$ runs the algorithm Update to update his access key $mak = (j, W_b)$. On receiving a random integer $l \leftarrow \mathbb{Z}_p^*$ from the AP, $\mathcal{B}$ chooses the tag base $(t_\iota = \delta_0, \check{t}_\iota = \check{\delta}_0, R_\iota)$, computes tag $(\Gamma, \check{\Gamma}) = (\Theta_0, \Phi^{(l x_b + l\check{t}_\iota + x_b)/(x_b^2 + x_b \check{t}_\iota)})$, and sends $(\Gamma, \check{\Gamma})$ to the AP with a simulation of the proof $Proof_2$, which can be done by using the simulator in the proof for $Proof_2$'s zero-knowledge property and resetting the random oracle. The AP ID_m and $\mathcal{B}$ perform the rest of the authentication protocol as specified in Section 4.2. $\mathcal{B}$ then outputs the transcript of the protocol.

From the transcript outputted by $\mathcal{O}_{QUERY}$, if $\mathcal{A}$ returns the bit b, then $\mathcal{B}$ decides that the tuple α is chosen from S_0. Otherwise, $\mathcal{B}$ decides that the tuple

α is chosen from S_1. Then if $\mathcal{A}$ can break the Anonymity property of the k-TAA scheme, then $\mathcal{B}$ can break the DBDHI assumption.

D.2 Proof of Theorem 1 (iii)

Suppose there exists a PPT adversary $\mathcal{A}$ breaking the Detectability property of our scheme, we show a PPT adversary $\mathcal{B}$ that can break the SDH assumption. Let $challenge = (R, R^z, \ldots, R^{z^q})$ be a tuple of the SDH assumption, where $z \leftarrow \mathbb{Z}_p^*$, $\mathcal{B}$'s challenge is to compute $(c, R^{1/(z+c)})$, where $c \in \mathbb{Z}_p$.

As $\mathcal{A}$ can break Detectability, at the end of the experiment with non negligible probability, $\mathcal{A}$ can be successfully authenticated by an honest AP $\mathcal{V}$ with access bound k for more than $k \times \#AG_{\mathcal{V}}$ times, where $\#AG_{\mathcal{V}}$ is the number of members in the AP's access group. As $Proof_2$ is zero-knowledge, for each of these successful authentication runs, $\mathcal{A}$ must have the knowledge of a tag base $(t_\iota, \check{t}_\iota, R_\iota)$, a member public key (a, S), a member secret key (x, v) and a member access key W. There are 3 possible cases:

- A member secret key (x, v) in $\mathcal{V}$'s access group is used for authentication for more than k times. As $\mathcal{V}$ provides only k tag bases, $\mathcal{A}$ must generate a new valid tag base $(t, \check{t}, R)$ to use with (x, v). Following arguments (which can't be shown due to space limitation) similar to the proof of Lemma 2 in [12], if $\mathcal{A}$ can generate a new valid tag base $(t, \check{t}, R)$, then the SDH assumption does not hold.
- No member secret key in $\mathcal{V}$'s access group is used for authentication for more than k times and $\mathcal{A}$ can generate a new member key pair $((a, S, \beta), (x, v))$ different from any member key pair of the whole group. Following arguments (which can't be shown due to space limitation) similar to the proof of Lemma 2 in [12], if this can be done, then the SDH assumption does not hold.
- No member secret key in $\mathcal{V}$'s access group is used for authentication for more than k times and $\mathcal{A}$ can generate a new member access key W for a group member, who is not in $\mathcal{V}$'s access group and has a member key pair $((a, S, \beta), (x, v))$. In this case, $\mathcal{B}$ simulates the GM, the users, the APs and randomly chooses an AP $\mathcal{V}$ with bound k and provides them to $\mathcal{A}$. $\mathcal{B}$ then runs the GKg algorithm to generate $gpk = (P, P_0, H', P_{pub}, \Phi, H, P_1, P_2)$ and $gsk = \gamma$ and runs the AKg algorithm for all APs, except $\mathcal{V}$. For $\mathcal{V}$, $\mathcal{B}$ selects $f, s' \leftarrow \mathbb{Z}_p^*$, and set $Q = R$, $Q_{pub} = R^z$, $Q'_{pub} = Q^{s'}$ and $V_0 = R^f$. The initial accumulated value is V_0 and $\mathcal{V}$'s keys are $((Q, Q_{pub}, Q'_{pub}), (z, s'))$, where $\mathcal{B}$ does not know z. With these capabilities, $\mathcal{B}$ can easily provide $\mathcal{A}$ access to simulations of the oracles $\mathcal{O}_{JOIN-GM}$, $\mathcal{O}_{AUTH-AP}$, $\mathcal{O}_{GRAN-AP}$, $\mathcal{O}_{REVO-AP}$ and $\mathcal{O}_{CORR-AP}$, except when $\mathcal{A}$ uses $\mathcal{O}_{CORR-AP}$ to corrupt $\mathcal{V}$, $\mathcal{B}$ fails and stops. Note that when $\mathcal{A}$ uses $\mathcal{O}_{GRAN-AP}$ or $\mathcal{O}_{REVO-AP}$ to ask $\mathcal{V}$ to grant access to or revoke access from a user, $\mathcal{B}$ can always use the tuple $challenge$ to compute the new accumulated value, as long as the number of users is less than q. As $\mathcal{B}$ randomly chooses $\mathcal{V}$, with non-negligible probability, $\mathcal{A}$ can be successfully authenticated by $\mathcal{V}$ for more than $k \times \#AG_{\mathcal{V}}$ times and generate a new member access key W for a group member, who is not in $\mathcal{V}$'s

access group and has a member key pair $((a, S, \beta), (x, v))$. Suppose the public keys of all members in AG_{ID} are $\{(a_i, \cdot, \cdot)\}_{i=1}^{m}$, then the current accumulated value of the AP is $V = R^{f\prod_{i=1}^{m}(a_i+z)}$, therefore $W = R^{f\prod_{i=1}^{m}(a_i+z)/(a+z)}$. From W and the tuple *challenge*, $\mathcal{B}$ can compute $R^{1/(a+z)}$ and thereby break the SDH assumption.

D.3 Proof of Theorem 1 (iv)

We show that if there exists a PPT adversary $\mathcal{A}$ breaking Exculpability for users in our scheme, then there exists a PPT adversary $\mathcal{B}$ breaking the Computational Bilinear Diffie-Hellman Inversion 2 assumption. Let $\mathbf{t} = (p, \mathbb{G}, \mathbb{G}_T, e, P') \leftarrow \mathcal{G}(1^\kappa)$ and suppose that $\mathcal{B}$ is given a challenge $\alpha = (P', P'^{w}, \ldots, P'^{(w^q)}, e(P', P')^{1/w})$ and $\mathcal{B}$ needs to compute w. $\mathcal{B}$ generates different $\delta_0, \check{\delta}_0, \delta_1, \check{\delta}_1, ..., \delta_{q-1}, \check{\delta}_{q-1} \leftarrow \mathbb{Z}_p^*$ and sets $x = w - \delta_0$. $\mathcal{B}$ simulates an instance of the dynamic k-TAA scheme and the oracles in the same way as simulations in the experiment of Anonymity proof, except that there is only one target user i and $e(P', P')^{1/w}$ is used instead of Λ. So we omit the description of simulations.

If $\mathcal{A}$ can break Exculpability for users, then Trace outputs i at the end of the experiment. That means there exist $(\Gamma_1, \check{\Gamma}_1, l_1, Proof)$ and $(\Gamma_2, \check{\Gamma}_2, l_2, Proof')$ in the log of an AP such that $\Gamma_1 = \Gamma_2$, $(\check{\Gamma}_1/\check{\Gamma}_2)^{1/(l_1-l_2)} = \beta(= \Phi^{1/x})$ and $Proof$ and $Proof'$ are valid. As $\mathcal{A}$ can only use $\mathcal{O}_{AUTH\_U}$ within the allowable numbers of times, not both $(\Gamma_1, \check{\Gamma}_1, l_1, Proof)$ and $(\Gamma_2, \check{\Gamma}_2, l_2, Proof')$ is created by $\mathcal{B}$ using the oracle.

In the case neither of them was created by $\mathcal{B}$ using the oracle, as $Proof_2$ is zero-knowledge, $\mathcal{A}$ must have the knowledge of $(x_1, t_1, \check{t}_1)$ and $(x_2, t_2, \check{t}_2)$ such that $(\Gamma_1, \check{\Gamma}_1) = (\Phi^{1/(x_1+t_1)}, \Phi^{(l_1x_1+l_1\check{t}_1+x_1)/(x_1^2+x_1\check{t}_1)})$; $(\Gamma_2, \check{\Gamma}_2) = (\Phi^{1/(x_2+t_2)}, \Phi^{(l_2x_2+l_2\check{t}_2+x_2)/(x_2^2+x_2\check{t}_2)})$; $\Gamma_1 = \Gamma_2$ and $\check{\Gamma}_1/\check{\Gamma}_2 = \Phi^{(l_1-l_2)/x}$. By converting all elements into exponents of Φ, one can compute x from $x_1, t_1, \check{t}_1, l_1, x_2, t_2, \check{t}_2, l_2$. Therefore, w is computable. By similar arguments for the case when one of $(\Gamma_1, \check{\Gamma}_1, l_1, Proof)$ or $(\Gamma_2, \check{\Gamma}_2, l_2, Proof')$ was created by $\mathcal{B}$ using the oracle, one can also find w.

D.4 Proof of Theorem 1 (v)

Suppose a PPT adversary $\mathcal{A}$ can break Exculpability for the GM in our scheme, we show that the SDH assumption does not hold. If Trace outputs GM at the end of the experiment, there exist $(\Gamma, \check{\Gamma}, l, Proof)$ and $(\Gamma', \check{\Gamma}', l', Proof')$ in the log of an AP such that $\Gamma = \Gamma'$, $(\check{\Gamma}/\check{\Gamma}')^{1/(l-l')} \notin LIST$ and $Proof$ and $Proof'$ are valid. As $Proof_2$ is zero-knowledge, $\mathcal{A}$ must have the knowledge of $(t, \check{t}, a, S, x, v)$ and $(t', \check{t}', a', S', x', v')$ such that $x+t = x'+t'$. If $t \neq t'$, then with non-negligible probability, either x or x' is not issued in the Join protocol with the GM; so a new valid member public key/secret key pair has been created without the GM. If $t = t'$, then $x = x'$. But $\Phi^{1/x} \notin LIST$, so x is not issued in the Join protocol with the GM; so a new valid member public key/secret key pair has also been created without the GM. Following arguments (which can't be shown

due to space limitation) similar to the proof of Lemma 2 in [12], if a new valid member public key/secret key pair can be created without the GM, then the SDH assumption does not hold.