k-times Anonymous Authentication with a Constant Proving Cost

Isamu Teranishi and Kazue Sako

NEC Corporation

Abstract. A k-Times Anonymous Authentication (k-TAA) scheme allows users to be authenticated anonymously so long as the number of times that they are authenticated is within an allowable number. Some promising applications are e-voting, e-cash, e-coupons, and trial browsing of contents. However, the previous schemes are not efficient in the case where the allowable number k is large, since they require both users and verifiers to compute $O(k)$ exponentiation in each authentication. We propose a k-TAA scheme where the numbers of exponentiations required for the entities in an authentication are independent of k. Moreover, we propose a notion of public detectability in a k-TAA scheme and present an efficient publicly verifiable k-TAA scheme, where the number of modular exponentiations required for the entities is $O(\log(k))$.

Keywords: k-times anonymous authentication, efficiency, public verifiability.

1 Introduction

1.1 Background

A k-Times Anonymous Authentication (k-TAA) scheme [TFS04,NN05] allows users to be authenticated anonymously so long as the number of times that they are authenticated is within an allowable number. The scheme not only offers a time restriction mechanism to well-known group signature schemes [CH91,ACJT00,BBS04,BSZ05,CG04,KY05], but provides stronger properties of anonymity and traceability. Regarding anonymity, users who are authenticated within the allowable number times can enjoy anonymity even from an authority, whereas in a group signature scheme users are always identifiable by the authority. Regarding traceability, any verifier can trace a malicious "over-time" user (that is, a user who exceeds the time restriction) from an authentication log in a k-TAA scheme, whereas in a group signature scheme it is only the authority who has this capability. There are many applications of the k-TAA scheme, such as e-voting [SK94,OMAFO99,DJ01,FS01, Neff01], e-cash [CP92,B93,AF96,CFT98,PBF99], e-coupon [OO98,NHS99], and trial browsing of contents [TFS04]. Moreover, these various application services can be offered based on a single secret key issued to users at the joining phase. For example, a user who paid an annual membership fee can download up to 1000 titles of digital music and 100 movie titles anonymously every year and can participate in anonymous questionnaires held every month.

We use the term "application providers" (APs) to refer to verifiers who wish to authenticate members and who want to restrict the number of times the members can use their service anonymously. In the previous example, the music downloading site is set to 1000, and the movie downloading site is set to 100, and monthly questionnaires site is set to 1.

A problem with the previous k-TAA schemes [TFS04,NN05] is that they require large computation when the number k becomes large, since $O(k)$ exponentiation is necessary for both users and AP.

1.2 Our Contributions

We propose, for the first time, a k-TAA scheme where the number of exponentiations required for users and AP for authentication is independent of k. The proposed scheme is constructed using a bilinear pairing [MSK02,BB04], and is secure under the SDH assumption [BB04], the DDHI assumption [BB04], and the random oracle assumption.

Moreover, we propose and formalize a stronger variant of the detectability requirement [TFS04] called the *public detectability*. It requires that anyone can verify that the AP indeed provided a fair limit to all users regarding the number of accesses to their service. For most applications such as e-coupon schemes and trial browsing, it is the APs that wish to restrict the number of times they offer service to a user, or else they will be paying for over-time users. So the property of the public detectability may not be necessary. However, in applications such as e-voting, one may want to publicly verify that the verifier has not accepted votes from the same user for more than a given number.

We also present an efficient and public detectable k-TAA scheme, where the number of exponentiations required for the user and verifier is $O(\log k)$. The scheme is secure under the same assumptions as those of the constant-cost scheme.

1.3 Key Ideas

We present here the ideas of the proposed schemes. First, we present the previous mechanism of detecting over-timed users [TFS04] at the cost of $O(k)$ exponentiations. We then show how this could be decreased to $O(\log k)$ and to $O(1)$. If an AP wishes to restrict the access time to be k, he publishes k public information items, namely, $r_1, \ldots, r_k$. In the authentication, a user picks one of the AP information items, say r_w, and sends a *tag* data ${r_w}^x$ using his secret key x. If a malicious user tries to be authenticated more than k times, the same tag ${r_{w'}}^x$ should appear in the authentication log and thus such a user will be detected. This is the mechanism to detect over-time users.

In the course of authentication a user needs to prove in zero knowledge that the tag is well-formed, that is, it is one of the public information items provided to his secret key. In the schemes of [TFS04,NN05], a user prove this using the 'OR proof', that is, a user proves that one of $\tau = {r_1}^x$, $\tau = {r_2}^x$, ..., or $\tau = {r_k}^x$ is satisfied. This resulted in the cost of $O(k)$ exponentiation.

In order to avoid the 'OR proof', we employ a deterministic function f_x to construct our tag. In the authentication, the user computes $\tau = f_x(ID_{\mathrm{AP}}||k, w)$

using his secret key x, sends it to the verifier and proves the inequality $1 \leq w \leq k$. The user can prove the inequality more efficiently by committing each bit of w. This is our first scheme with the public detectability property at the cost of only $O(\log k)$ exponentiation.

In our second scheme, the AP publishes signatures $\mathsf{Sig}(1), \ldots, \mathsf{Sig}(k)$ in advance. In the authentication, the user computes $\tau = f_x(ID_{\mathrm{AP}}||k, w)$ but proves the knowledge of a signature $\mathsf{Sig}(w)$, instead of proving the inequality regarding w. Since only k signatures $\mathsf{Sig}(1), \ldots, \mathsf{Sig}(k)$ are published, it indirectly ensures that $1 \leq w \leq k$. This resulted in the cost of only $O(1)$ exponentiation. However, this is not publicly detectable since a malicious AP may secretly reveal $\mathsf{Sig}(w)$ for $w > k$.

Based on the ideas above, we sought the best choice for the function f. We observed that a weakened pseudorandom function f is sufficient for our purpose and were able to choose efficiently computable f.

1.4 Related Works

E-cash schemes are similar to k-TAA schemes in the sense that they issue e-coins, or identification tokens, that can be used k times. A major difference between e-cash schemes and k-TAA schemes lies in the meaning of k, and who determines it.

In an e-cash scheme, the number k refers to the upper bound of the number of times the token issued by the Bank can be used. The Bank specifies k at the withdrawal phase, and the token can be spent in any shops. In contrast, in k-TAA schemes, Group Manager issues identification token at joining phase, but the use of this token is not limited. Instead, in k-TAA schemes, we want to limit the number of times this token is used in each shop, or in our term, each Application Provider. So it will be each AP that determines the number k, which is the upper bound of the number of times a user can use the token to received services from AP. There can be multiple APs and each of them can determine the upper bound independently. So a same token can be used at most say 100 times to Provider-1, 20 times to Provider-2 , and maybe once to Provider-3.

Having said the difference in the model, the techniques used in e-cash schemes and in our schemes is very similar. Independent to our work, Camenisch et. al. presented an e-cash scheme [CHL05] using similar ideas in our first scheme. The difference is in techniques in showing the inequality of $1 \leq w \leq k$. Although the smart use of Boudot scheme [B00] makes their scheme more efficient than our first scheme, it still requires number of exponentiations to be dependent of k, namely $O(\log k)$.

2 Definition of k-TAA scheme

2.1 Modified Points

Our definition is based on that of Teranishi et.al. [TFS04] with generalizations of allowing an AP to publish its own public information besides its ID and its allowable number k.

The generalization requires us to modify the previous definition in two other points, which are concerned with the "public tracing" algorithm. This is an algorithm which enables anyone to identify a user authenticated more times than the allowable number. In the case where the identification of the user fails, it also enables anyone to know why it does so.

We next describe our modifications. Since we allow an AP to publish its public information, we add a new type of output "AP", which means "the identification fails since the AP publishes a maliciously generated public information items (or behaves maliciously in an authentication)", to a public tracing procedure. We also add new security requirements, called the exculpability for APs. This requires that the public tracing algorithm outputs AP only if the AP is dishonest.

We note that Nguyen and Naini [NN05] also adopt the definition which allows an AP to publish its own public information, but they do not adopt the other two modifications.

2.2 Model

Three types of entities take part in the model, namely, the *group manager* (*GM*), *users*, and *AP*s. The k-TAA scheme comprises the three algorithms *GM setup* ($\mathsf{GM\text{-}Setup}$), *AP setup* ($\mathsf{AP\text{-}Setup}$), and *public tracing* (trace) and two pairs of interactive protocols *joining* ($\mathsf{Join} = (\mathsf{Join\text{-}U}, \mathsf{Join\text{-}GM})$), and *authentication* ($\mathsf{Auth} = (\mathsf{Proof}, \mathsf{Verify})$). In the definitions below, κ is a security parameter.

$\mathsf{GM\text{-}Setup}$: The GM executes $\mathsf{GM\text{-}Setup}$ on inputting 1^κ and obtains a *GM public key/GM secret key* pair $(\mathsf{gpk}, \mathsf{gsk})$. Then it publishes gpk.

$\mathsf{AP\text{-}Setup}$: Each AP v determines the *allowable number* $k = k_v$, which indicates how many times the AP v allows each user to access. The AP v executes the $\mathsf{AP\text{-}Setup}$ on inputting its ID v and k, and obtains an *AP public information* api.

$\mathsf{Join} = (\mathsf{Join\text{-}U}, \mathsf{Join\text{-}GM})$: A user who wants to be a *group member* executes a Join protocol with the GM. The user and the GM execute $\mathsf{Join\text{-}U}$ and $\mathsf{Join\text{-}GM}$ respectively. The user's ID and gpk are input to both the $\mathsf{Join\text{-}U}$ and $\mathsf{Join\text{-}GM}$, and gsk is input only to the $\mathsf{Join\text{-}GM}$. The aims of the protocol are to add new members to the group and to generate new *member public key/secret key pair* $(\mathsf{mpk}, \mathsf{msk})$. If the Join protocol is successful, the user obtains both mpk and msk, and the GM obtains only mpk.

The member public key mpk comprises two parts. One part mck is called the *member certificate key* and the other part mik is called the *member identification key*. The key mck is a certificate which proves that the user is a member of the group. The key mik is added to the public list List along with the user's ID and will be used in order to identify the user.

$\mathsf{Auth} = (\mathsf{Proof}, \mathsf{Verify})$: An AP executes an Auth protocol with a user who wants to access the AP. The user and the AP execute Proof and Verify respectively. The public information $(\mathsf{gpk}, v, k, \mathsf{api})$ are input to both Proof and Verify, and msk is input only to Proof. If the protocol is successful, the AP records the data sent by the user in its *authentication log* Log, and outputs accept or reject. Here accept means that "the user is a group member and has not accessed the AP more times than the allowable number k".

trace : Anyone can execute trace algorithm using only public information (gpk, List, v, k, api) and the authentication log Log of an AP. The output of trace algorithm is either some user's ID u, "GM", "AP", or "NoOne". These four types of output respectively mean "the algorithm finds a malicious user u who is authenticated by the AP more times than the allowable number", "the algorithm finds that the GM published maliciously generated public information (gpk, List)", "the algorithm finds that the AP published maliciously generated (api, k, Log) or behaves maliciously in an authentication", and "the algorithm could not find any malicious entity".

We note that an AP can always mask a malicious user and generate Log, such that trace algorithm with input Log outputs "NoOne". That is, the AP can delete entries of the over times users.

2.3 Informal Definition of Requirements

A *secure* k-TAA scheme has to satisfy the following requirements:

Correctness: An honest group member is always accepted in an authentication by an honest AP.
Total Anonymity: No one is able to identify any authenticated member, or decide whether two accepted authentication protocols are performed by the same group member or not, if the authenticated user(s) has followed the authentication protocol within the allowed number of times per AP. These are satisfied even if all other users, the GM, and all APs collude with one another.
Exculpability for Users: trace algorithm does not output the ID of an honest user who is authenticated within the allowed number of times. This is satisfied even if all other users, the GM, and all APs collude with one another.
Exculpability for the GM: trace algorithm does not output "GM" if the GM is honest. This is satisfied even if all users and all APs collude with one another.
Exculpability for APs: trace algorithm using an honest AP's authentication log does not output "AP". This is satisfied even if the GM, all users and all other APs collude with one another.
Detectability: trace algorithm using an honest AP's authentication log does not output "NoOne", if a colluding subset of group members has been authenticated more than kn times. Here k is the allowable number set by the AP and n is the number of colluders.

We stress that the detectability property is satisfied only if the AP is honest. For most applications such as e-coupon schemes and trial browsing, the AP does not have to be honest, since it is the AP itself who wishes to limit the number of times to serve a user. However in applications such as e-voting, one wants to publicly verify that the AP has not accepted votes from the same user for more than a given number. In order to meet such applications, we newly introduce the stronger detectability notion, *public detectability*:

Public Detectability: trace algorithm using an honest or dishonest AP's authentication log does not output NoOne, if the log contains more than kn malicious entries. Here k is the allowable number set by the AP and n is the number of the group members. This is satisfied even if every user and every AP collude with one another.

2.4 Formal Definition of Requirements

We modify the experiments for defining the requirements of the previous paper [TFS04] in order to suit the modification described in 2.1, but our experiments are essentially the same as those of [TFS04]. The major modification is that we introduce the oracles $\mathcal{O}_{\mathsf{AP\text{-}Setup}}$ and $\mathcal{O}_{\mathsf{VList}}$. Here the former is the oracle which executes $\mathsf{AP\text{-}Setup}$ honestly, and latter is the oracle which manages the public list VList of a pair of AP's IDs v, the allowable number k set by the AP, and its public information api. We will describe the details of $\mathcal{O}_{\mathsf{VList}}$ later.

The experiments of our version of the total anonymity and the exculpability for users and for the GM are the same as those of [TFS04], except that an adversary is allowed to access the oracle $\mathcal{O}_{\mathsf{VList}}$. Moreover, the definition of the experiment for defining the detectability is also the same as that of [TFS04], except an adversary is allowed to access oracle $\mathcal{O}_{\mathsf{VList}}$ and the oracle $\mathcal{O}_{\mathsf{AP\text{-}Setup}}$.

Before we describe the experiments, we first describe what $\mathcal{A}$ can do when it colludes with GM, users, and AP respectively.

- If $\mathcal{A}$ colludes with the GM, it can maliciously execute $\mathsf{GM\text{-}Setup}$ and $\mathsf{Join\text{-}GM}$.
- If $\mathcal{A}$ colludes with a user, it can execute $\mathsf{Join\text{-}U}$ and Proof maliciously on behalf of the user.
- If $\mathcal{A}$ colludes with an AP, it can execute $\mathsf{AP\text{-}Setup}$ and Verify maliciously on behalf of the AP.

We next describe the oracles. Let $\mathcal{O}_{\mathsf{Join\text{-}GM}}$ be the oracle which executes $\mathsf{Join\text{-}GM}$ procedures honestly. Let $\mathcal{O}_{\mathsf{Join\text{-}U}}$ and $\mathcal{O}_{\mathsf{Proof}}$ be oracles which execute $\mathsf{Join\text{-}U}$ and Proof procedures on behalf of honest users. Similarly, let $\mathcal{O}_{\mathsf{AP\text{-}Setup}}$ and $\mathcal{O}_{\mathsf{Verify}}$ be the oracles which execute $\mathsf{AP\text{-}Setup}$ and Verify on behalf of honest verifiers.

We also introduce the *list oracle* $\mathcal{O}_{\mathsf{List}}(X, \cdot)$ [TFS04], which manages the public list List of a pair of user's IDs and his member identification key mik, and allows $\mathcal{A}$ to read List. Moreover, the oracle also manages the set X of IDs of entities who collude with an adversary $\mathcal{A}$, and it allows $\mathcal{A}$ to write an (honestly or dishonestly generated) pair (u, mik) if $\mathcal{A}$ colluded with the user u. The list oracle also allows $\mathcal{A}$ to delete entries of List if it colludes with the GM.

We also introduce the new list oracle $\mathcal{O}_{\mathsf{List}}$, which manages the public list VList of pairs of AP's IDs and their public information. The definition of $\mathcal{O}_{\mathsf{VList}}$ is quite similar to that of the original $\mathcal{O}_{\mathsf{List}}$. However, $\mathcal{O}_{\mathsf{VList}}$ does not allow the GM to delete the data of List. This is because, even in the actual scenario, VList is managed not by the GM, but by some trusted party (such as a Certified Authority of a PKI). Therefore the GM cannot delete entries of VList. See the full paper for the formal definition of $\mathcal{O}_{\mathsf{List}}$ and $\mathcal{O}_{\mathsf{VList}}$.

$\mathcal{A}$ is allowed to access oracles only sequentially. We describe when $\mathcal{A}$ is allowed to access the oracles.

- $\mathcal{A}$ is allowed to access the list oracles $\mathcal{O}_{\mathsf{List}}$ and $\mathcal{O}_{\mathsf{VList}}$.

– If $\mathcal{A}$ does not collude with the GM, it is allowed to access $\mathcal{O}_{\mathsf{Join\text{-}GM}}$.
– If $\mathcal{A}$ does not collude with a user u, it is allowed to access $\mathcal{O}_{\mathsf{Join\text{-}U}}$ and $\mathcal{O}_{\mathsf{Proof}}$. Here these oracles take roles of the user u.
– If $\mathcal{A}$ does not collude with an AP v, it is allowed to access $\mathcal{O}_{\mathsf{AP\text{-}Setup}}$ and $\mathcal{O}_{\mathsf{Verify}}$. Here these oracles take roles of the AP v.

We now describe the experiments for defining the requirements. Figure 1 describes the security experiments formally. Here κ is a security parameter, $(\mathsf{gpk}, \mathsf{gsk})$ is a GM public key/secret key pair, mik is a member identification key, and api is AP public information.

—$\mathbf{Exp}_{\mathcal{A}}^{\text{anon-}(u_1,u_2,\beta)}(\kappa)$—
$(\mathsf{gpk}, v, k, \mathsf{api}, \mathsf{St}) \leftarrow \mathcal{A}(1^\kappa)$
$\beta' \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{anon}}(\cdot)}(\mathsf{St})$
If $(N_1, N_2 \leq k$ and $\beta = \beta')$ Return Win.
Return Lose.

—$\mathbf{Exp}_{\mathcal{A}}^{\text{excul-}u_1}(\kappa)$—
$(\mathsf{gpk}, \mathsf{St}) \leftarrow \mathcal{A}(1^\kappa)$.
$(v, k, \mathsf{api}, \mathsf{Log}) \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{excul}\text{-}u_1}(\cdot)}(\mathsf{St})$.
Return $\mathsf{trace}^{\mathcal{O}_{\mathsf{List}}(\emptyset,\cdot)}(\mathsf{gpk}, v, k, \mathsf{api}, \mathsf{Log})$.

—$\mathbf{Exp}_{\mathcal{A}}^{\text{detect}}(\kappa)$—
$(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{GM\text{-}Setup}(1^\kappa)$
$\mathcal{A}^{\mathsf{O}_{\mathsf{detect}}(\cdot)}(1^\kappa, \mathsf{gpk})$.
If $({}^{\exists}(v,k) \in \mathsf{VList}$ s.t. $\#\mathsf{Log}_{v,k} > k \cdot \#\mathsf{List})$
  Return $\mathsf{trace}^{\mathcal{O}_{\mathsf{List}}(\emptyset,\cdot)}(1^\kappa, \mathsf{gpk}, v, k, \mathsf{api}, \mathsf{Log}_{v,k})$.
Return $\perp$.

—$\mathbf{Exp}_{\mathcal{A}}^{\text{excul-GM}}(\kappa)$—
$(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{GM\text{-}Setup}(1^\kappa)$
$(v, k, \mathsf{api}, \mathsf{Log}) \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{excul\text{-}GM}}(\cdot)}(\kappa)$.
Return $\mathsf{trace}^{\mathcal{O}_{\mathsf{List}}(\emptyset,\cdot)}(\mathsf{gpk}, v, k, \mathsf{api}, \mathsf{Log})$.

—$\mathbf{Exp}_{\mathcal{A}}^{\text{excul-AP-}(v,k)}(\kappa)$—
$\mathsf{api} \leftarrow \mathsf{AP\text{-}Setup}^{\mathcal{O}_{\mathsf{VList}}(\{v\},\cdot)}(1^\kappa, v, k)$.
$(\mathsf{gpk}, \mathsf{St}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{VList}}(\{v\}^c,\cdot)}(1^\kappa, v, k)$.
$\mathcal{A}^{\mathsf{O}_{\mathsf{excul\text{-}AP\text{-}}(v,k)}(\cdot)}(\mathsf{St})$.
Return $\mathsf{trace}^{\mathcal{O}_{\mathsf{List}}(\emptyset,\cdot)}(\mathsf{gpk}, v, k, \mathsf{api}, \mathsf{Log})$.

—$\mathbf{Exp}_{\mathcal{A}}^{\text{pub-detect}}(\kappa)$—
$(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{GM\text{-}Setup}(1^\kappa)$
$(v, k, \mathsf{api}, \mathsf{Log}) \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{pub\text{-}detect}}(\cdot)}(1^\kappa, \mathsf{gpk})$.
If $(\#\mathsf{Log} > k \cdot \#\mathsf{List})$
  Return $\mathsf{trace}^{\mathcal{O}_{\mathsf{List}}(\emptyset,\cdot)}(1^\kappa, \mathsf{gpk}, v, k, \mathsf{api}, \mathsf{Log})$.
Return $\perp$.

Oracles:
$\mathsf{O}_{\mathsf{anon}}(\cdot) = (\mathcal{O}_{\mathsf{List}}(\{u_1, u_2\}^c, \cdot), \mathcal{O}_{\mathsf{VList}}(\emptyset^c, \cdot), \mathcal{O}_{\mathsf{JOIN\text{-}U}}(\mathsf{gpk}, \cdot), \mathcal{O}_{\mathsf{Proof}}(\mathsf{gpk}, \cdot),$
$\mathcal{O}_{\mathsf{Query}}(\beta, \mathsf{gpk}, (u_1, u_2), (v, k, \mathsf{api}), (\cdot,\cdot)))$
$\mathsf{O}_{\mathsf{excul}\text{-}u_1}(\cdot) = (\mathcal{O}_{\mathsf{List}}(\{u_1\}^c, \cdot), \mathcal{O}_{\mathsf{VList}}(\emptyset^c, \cdot)\mathcal{O}_{\mathsf{JOIN\text{-}U}}(\mathsf{gpk}, \cdot), \mathcal{O}_{\mathsf{Proof}}(\mathsf{gpk}, \cdot))$
$\mathsf{O}_{\mathsf{excul\text{-}GM}}(\cdot) = (\mathcal{O}_{\mathsf{List}}(\{\mathsf{GM}\}^c, \cdot), \mathcal{O}_{\mathsf{VList}}(\emptyset^c, \cdot), \mathcal{O}_{\mathsf{JOIN\text{-}GM}}(\mathsf{gpk}, \mathsf{gsk}, \cdot))$
$\mathsf{O}_{\mathsf{excul\text{-}AP\text{-}}(v,k)}(\cdot) = (\mathcal{O}_{\mathsf{List}}(\emptyset^c, \cdot), \mathcal{O}_{\mathsf{VList}}(\{v\}^c, \cdot), \mathcal{O}_{\mathsf{Verify}}(\mathsf{gpk}, (v, k, \mathsf{api}), \cdot))$
$\mathsf{O}_{\mathsf{detect}}(\cdot) = (\mathcal{O}_{\mathsf{List}}(\{\mathsf{GM}\}^c, \cdot), \mathcal{O}_{\mathsf{VList}}(\emptyset^c, \cdot), \mathcal{O}_{\mathsf{Join\text{-}GM}}(\mathsf{gpk}, \mathsf{gsk}, \cdot), \mathcal{O}_{\mathsf{AP\text{-}Setup}}(1^\kappa, \cdot, \cdot), \mathcal{O}_{\mathsf{Verify}}(\mathsf{gpk}, \cdot, \cdot)$
$\mathsf{O}_{\mathsf{pub\text{-}detect}}(\cdot) = (\mathcal{O}_{\mathsf{List}}(\{\mathsf{GM}\}^c, \cdot), \mathcal{O}_{\mathsf{VList}}(\emptyset^c, \cdot), \mathcal{O}_{\mathsf{Join\text{-}GM}}(\mathsf{gpk}, \mathsf{gsk}, \cdot))$

Comments:
1. To simplify, we abbreviate the hash oracle $\mathcal{O}_{\mathsf{Hash}}$.
2. In the experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{anon-}(u_1,u_2,\beta)}(\kappa)$, N_i is the total number of times $\mathcal{O}_{\mathsf{JOIN\text{-}U}}$ and $\mathcal{O}_{\mathsf{Query}}$ executes Proof using using a public key/secret key pair of user $u_{\beta \oplus d+1}$ and an APs public information (v, k, api).
3. In the definition of $\mathbf{Exp}_{\mathcal{A}}^{\text{detect}}(\kappa)$, $\mathsf{Log}_{v,k}$ is the log of $\mathcal{O}_{\mathsf{Verify}}$ on the behalf of the AP v with the allowable number k.
4. In the definition of $\mathbf{Exp}^{\text{excul-AP-}(v,k)}$, Log is the log of $\mathcal{O}_{\mathsf{Verify}}$.

Fig. 1. The experiments.

Total Anonymity: In advance, two target users u_1 and u_2 are determined, and a secret number $\beta \in \{0, 1\}$ is selected randomly. An adversary $\mathcal{A}$ is allowed to collude with the GM, all APs, and all users except target users u_1 and u_2.

First, $\mathcal{A}$ determines and publishes the group public key gpk, an AP's ID v, the allowable number k of the AP, and the AP's public information api. Next, $\mathcal{A}$ maliciously executes the Join and Auth protocols with $\mathcal{O}_{\mathsf{Join\text{-}U}}$ and $\mathcal{O}_{\mathsf{Proof}}$. These oracles execute protocols on the behalf of the target users.

Moreover, $\mathcal{A}$ is allowed to access the *query oracle* $\mathcal{O}_{\mathsf{Query}}(\beta, \mathsf{gpk}, (u_1, u_2), (v, k, \mathsf{api}), (\cdot, \cdot))$. We give the definition of the query oracle. The oracle executes Proof algorithm on the behalf of a target user, but does not disclose which target user the oracle takes the role of. More precisely, if $\mathcal{A}$ sends (d, M) to oracle $\mathcal{O}_{\mathsf{Query}}(\beta, \mathsf{gpk}, (u_1, u_2), (v, k, \mathsf{api}), (\cdot, \cdot))$, the oracle regards M as data sent by a user and executes Proof using a public key/secret key pair of user $u_{\beta \oplus d+1}$ and an APs public information (v, k, api). $\mathcal{A}$ is allowed to execute the Auth protocol with the query oracle once only for each $d \in \{0, 1\}$. If $\mathcal{A}$ requires for the oracle to execute the Auth protocol for the same d twice, $\mathcal{O}_{\mathsf{Query}}$ returns $\perp$.

In the experiment, $\mathcal{A}$ is not allowed to authenticate the target user u_i more than k times. This is because a k-TAA scheme provides anonymity to users only if a user has been authenticated less than the allowed number of times. More precisely, let N_i be the total number of times $\mathcal{O}_{\mathsf{Join\text{-}U}}$ and $\mathcal{O}_{\mathsf{Query}}$ execute Proof using a public key/secret key pair of user u_i. Then $\mathcal{A}$ must preserve $N_1, N_2 \leq k$.

The aim of $\mathcal{A}$ is to determine whether $\beta = 1$ or not. $\mathcal{A}$ wins if $N_1, N_2 \leq k$ is satisfied and $\mathcal{A}$ succeeds in outputting β.

Exculpability for Users: In the experiment for defining the exculpability for users, a target user u is fixed in advance. $\mathcal{A}$ is allowed to collude with all entities except the target user u. If $\mathcal{A}$ succeeds in computing the log with which the public tracing procedure outputs the ID u of the target user, it wins.

Exculpability for GM: $\mathcal{A}$ is allowed to collude with all entities except the GM. If $\mathcal{A}$ succeeds in computing the log with which the public tracing procedure outputs "GM", it wins.

Exculpability for APs: A target AP v is fixed in advance. $\mathcal{A}$ is allowed to collude with all entities except the target AP v. Let Log be the authentication log of $\mathcal{O}_{\mathsf{Verify}}$. If a public tracing procedure using Log outputs "AP", $\mathcal{A}$ win. We stress that not $\mathcal{A}$ but $\mathcal{O}_{\mathsf{Verify}}$ outputs Log in this experiment, although adversaries of the other two exculpability properties are allowed to output Log themselves.

Detectability: $\mathcal{A}$ is allowed to collude with all users. If $\mathcal{A}$ succeeds in being accepted by some AP in more than kn authentications, $\mathcal{A}$ wins. Here, k is the number of times the AP allows access for each user, and n is the number of users who collude with $\mathcal{A}$.

Public Detectability: $\mathcal{A}$ is allowed to collude with all users and all APs. $\mathcal{A}$ wins if $\mathcal{A}$ succeeds in outputting a tuple $(v, k, \mathsf{api}, \mathsf{Log})$ satisfying both of the following conditions: (1) the authentication Log contains more than $k \cdot \#\mathsf{List}$ elements and (2) a public tracing procedure using $(v, k, \mathsf{api}, \mathsf{Log})$ outputs NoOne.

Definition 1 We say a k-TAA scheme satisfies the *total anonymity*, *exculpability for users*, *exculpability for GM*, *exculpability for APs*, *detectability* and *public detectability* properties if no adversary can win with a non negligible advantage in the experiments for defining these requirements. More precisely, we say a k-TAA scheme satisfies these requirements if $|\Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{anon-}(u_1,u_2,0)}(\kappa) = \mathsf{Win}) - \Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{anon-}(u_1,u_2,1)}(\kappa) = \mathsf{Win})|$, $\Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{excul}-u_1}(\kappa) = u_1)$, $\Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{excul-GM}}(\kappa) = \mathsf{GM})$, $\Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{excul-AP-}(v,k)}(\kappa) = \mathsf{AP})$, $\Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{detect}}(\kappa) = \mathsf{NoOne})$ and $\Pr(\mathbf{Exp}_{\mathcal{A}}^{\text{pub-detect}}(\kappa) = \mathsf{NoOne})$ are negligible for security parameter κ, for all $(\mathcal{A}, u_1, u_2)$, $(\mathcal{A}, u_1)$, $\mathcal{A}$, $(v, k, \mathcal{A})$, $\mathcal{A}$ and $\mathcal{A}$ respectively.

We say a k-TAA scheme is *secure* if it satisfies the first five requirements.

3 Proposed Schemes

We propose two schemes. Authentications of the first and the second schemes require computing $O(\log k)$ or $O(1)$ exponentiations respectively. Although the first scheme is less efficient than the second scheme, only the first scheme satisfies the public detectability property. As in the previous schemes [TFS04,NN05], our proposed schemes are based on a group signature scheme. We adopt the Furukawa-Imai scheme [FI05], since it is one of the most efficient group signature schemes. The GM setup and the joining procedures of our two schemes are similar to those of [FI05].

3.1 Notations

Let κ be a security parameter. Let $(\mathcal{G}, \mathcal{H}, \mathcal{T}, q, \langle\cdot,\cdot\rangle, \phi)$ be a bilinear pairing tuple, that is, a tuple satisfying the following properties: (1) q is a prime number whose bit length is κ, (2) $\mathcal{G}, \mathcal{H}$, and $\mathcal{T}$ are cyclic groups of order q, (3) ϕ is a polynomial time computable homomorphism from $\mathcal{H}$ to $\mathcal{G}$, (ϕ is called *distorsion map*), (4) $\langle\cdot,\cdot\rangle$ is a polynomial time computable mapping from $\mathcal{G} \times \mathcal{H}$ to $\mathcal{T}$, (5) for all $(a, b) \in \mathcal{G} \times \mathcal{H}$, if $\langle a, b\rangle = 1$ is satisfied, then $a = b = 1$ is satisfied, and (6) for all $a \in \mathcal{G}$, $b \in \mathcal{H}$, and $x, y \in \mathbb{Z}_q$, $\langle a^x, b^y\rangle = \langle a, b\rangle^{xy}$ is satisfied.

Let $\mathcal{U}$ be a group on which the DDH problem is hard and whose order is the same as that of $\mathcal{T}$. Although we can set $\mathcal{U}$ to $\mathcal{T}$ itself, the Furukawa-Imai scheme and our schemes become more efficient if we set $\mathcal{U}$ to an elliptic curve on which a pairing is not defined.

3.2 First Scheme

Let Hash denote a full domain hash function onto set $\mathcal{U}^2$. For a bit string X, let (g_X, h_X) denote $\mathsf{Hash}(X)$. For $x \in \mathbb{Z}_q$, we set $f_x^\kappa : \{0,1\}^* \times \mathbb{Z}_q \to \mathcal{U}$ to

$$f_x^\kappa : (X, w) \mapsto {g_X}^w {h_X}^{1/(x+w)}.$$

$\mathsf{GM\text{-}Setup}$: The $\mathsf{GM\text{-}Setup}$ generates and outputs a GM public key $\mathsf{gpk} = (a_0, a_1, a_2, b, b')$ and a GM secret key $\mathsf{gsk} = y$ following [FI05]. That is, the algorithm selects $a_0, a_1, a_2 \in \mathcal{G}$, $b \in \mathcal{H}$, and $y \in \mathbb{Z}_q$ randomly, and computes $b' = b^y$.

Join: The Join protocol generates a member certificate key $\mathsf{mck} = (A, e)$, a member identification key $\mathsf{mik} = f_x^\kappa(0, 0)$, and the member secret key $\mathsf{msk} = (x, r)$, following [FI05]. These keys satisfy the equation $\langle a_0 {a_1}^x {a_2}^r, b\rangle = \langle A, b^e b'\rangle$. The algorithm outputs $(\mathsf{mck}, \mathsf{mik}, \mathsf{msk})$ for users and outputs only mck for the GM.

More precisely, the GM and a user perform as described in Figure 2, where u is the ID of the user, mik is the member identification key of the user, and List is the list of the user's ID and his member identification key. Then the user checks that $\langle a_0 {a_1}^x {a_2}^r, b\rangle = \langle A, b^e b'\rangle$ is satisfied.

User		GM
$x, r' \leftarrow \mathbb{Z}_q, A' \leftarrow {a_1}^x {a_2}^{r'}$		
$\mathsf{mik} \leftarrow h \leftarrow f_x^{\kappa}(0,0)$		
Add (u, mik) to List.	$\xrightarrow{A', \mathsf{mik}}$	Check that (u, mik) is in List.
Proof the validity of $(A', \mathsf{mik}))$	$\longleftrightarrow$	Verify the proof.
		$r'', e \leftarrow \mathbb{Z}_q$
$r \leftarrow r' + r'' \bmod q$	$\xleftarrow{((A,e),r'')}$	$A \leftarrow ({a_0}^{1/(y+e)} A' a_2^{r''/(y+e)})$

Fig. 2. Joining of the First Scheme

Auth: In w-th authentication, a user and an AP first perform as in Figure 3. Here pf is a proof of knowledge of $((A^*, e^*), (x^*, r^*), w^*)$ satisfying the following conditions: (T1): $\langle a_0 {a_1}^{x^*} {a_2}^{r^*}, b\rangle = \langle A, b^{e^*} b'\rangle$, (T2): $(\tau, \hat{\tau}) = (f_{x^*}^{\kappa}(v||k, w^*), f_{x^*}^{\kappa}(0,0)^{\ell} f_{x^*}^{\kappa}(v||k, -w^*))$, and (T3): $1 \leq w^* \leq k$.

User		AP
	$\xleftarrow{\ell}$	$\ell \leftarrow \mathbb{Z}_q$
$(\tau, \hat{\tau}) \leftarrow (f_x^{\kappa}(v\|\|k, w), \mathsf{mik}^{\ell} \cdot f_x^{\kappa}(v\|\|k, -w))$		
$\mathsf{pf} \leftarrow$ (Validity proof of $(\tau, \hat{\tau})$)	$\xrightarrow{((\tau,\hat{\tau}),\mathsf{pf})}$	Verify pf.

Fig. 3. Authentication

The AP next executes the following procedures. Let Log be the AP's authentication log. If $\mathsf{Ver}(\mathsf{pf}) = \mathsf{accept}$ and $\tau \notin \mathsf{Log}$ are satisfied, add $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ to Log and output accept. If $\mathsf{Ver}(\mathsf{pf}) = \mathsf{accept}$ but $\tau \in \mathsf{Log}$ are satisfied, add $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ to Log but output reject. Otherwise, add no data to Log and output reject.

trace: From Log, the trace algorithm searches entries $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ and $(\tau', \hat{\tau}', \ell', \mathsf{pf}')$ satisfying $\tau = \tau'$. We first consider the case where such entries exist. Then the algorithm verifies pf and pf'. If $\ell = \ell'$ is satisfied, output AP and stop. Otherwise, the algorithm computes $\mathsf{mik} = (\hat{\tau}/\hat{\tau}')^{1/(\ell-\ell')}$ and searches the ID corresponding with mik from List. If there is such an ID, the algorithm outputs the ID and stop, otherwise outputs GM and stops.

We next consider the case where there exists no pair of entries $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ and $(\tau', \hat{\tau}', \ell', \mathsf{pf}')$ satisfying $\tau = \tau'$ in Log. Then in order to check that the AP added invalid entries to the Log, the algorithm verifies all proofs in Log. If some proof is invalid, output AP and stops. If all proofs in Log are valid, the algorithm outputs NoOne and stops.

We note that one must verify all proofs in trace in order to ensure the public detectability. One is not required to verify this if only normal detectability is required.

3.3 Second Scheme

By modifying the first scheme, we construct the second k-TAA scheme such that the numbers of exponentiations in an authentication is $O(1)$. In order to reduce the computational cost of an authentication, we use a signature scheme. In our second scheme, AP publishes signatures $\mathsf{Sig}(1)$, ..., $\mathsf{Sig}(k)$ in its setup. In the authentication, the user computes $(\tau, \hat{\tau})$ as in the first scheme, but proves the knowledge of a signature $\mathsf{Sig}(w)$, instead of proving the inequality $1 \leq w \leq k$. Since only k signatures $\mathsf{Sig}(1)$, ..., $\mathsf{Sig}(k)$ are published, it indirectly ensures that $1 \leq w \leq k$. This resulted in the cost of only $O(1)$ exponentiation. However, this is not publicly detectable since a malicious AP may secretly reveal $\mathsf{Sig}(w)$ for $w > k$.

We use the following Boneh-Boyen signature scheme (SGen, Sig,SVer) [BB04] in order to construct our second scheme. Here SGen, Sig, and SVer are respectively the key generation, the signing, and the verification algorithms:

SGen: The algorithm selects $s \in \mathcal{G}$, $t \in \mathcal{H}$ and $z \in \mathbb{Z}_q$ randomly and computes $t' = t^z$. Then it outputs the public key (s, t, t') and the corresponding secret key z.

Sig: If a message $w \in \mathbb{Z}_q$ is input, the algorithm output a signature $S = s^{1/(z+w)}$ on w.

SVer: The algorithm accepts (w, S) if and only if $\langle S, t^w t' \rangle = \langle s, t \rangle$ is satisfied.

We now describe our second scheme. Let f_x^κ be the deterministic function described in 3.2.

$\mathsf{GM\text{-}Setup}$ and Join : These are the same as those of the first scheme.

$\mathsf{AP\text{-}Setup}$: Let v be an AP's ID. The AP v determines the allowable number k. Generate a public key/private key pair $(\mathsf{spk}, \mathsf{ssk}) = \mathsf{SGen}(1^\kappa)$ of the signature scheme [BB04]. Then compute signatures $S_w = \mathsf{Sig}_{\mathsf{spk},\mathsf{ssk}}(w)$ on w for all $w = 1, \ldots, k$. The AP public information is $\mathsf{api} = (\mathsf{spk}, \{S_w\})$. Add (v, k, api) to the AP public information list VList.

Auth: The only difference from the first scheme is what the user proves. In the second scheme, pf is the proof of knowledge of $(\mathsf{mck}^*, (x^*, r^*), w^*, S^*)$ which satisfies (T1) and (T2) of 3.2 and (T3'): $\mathsf{SVer}_{\mathsf{spk}}(w^*, S^*) = \mathsf{accept}$.

trace: Search $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ and $(\tau', \hat{\tau}', \ell', \mathsf{pf}')$, satisfying $\tau = \tau'$ from Log. If there exist such entries, subsequent procedures are the same as those of the first scheme. If there exist no such entries, output NoOne and stop.

We next show that the second scheme does not satisfy the public detectability property. Indeed, an AP is able to generate any number of signatures, and therefore a colluding subset of the AP and a user is able to generate any number of entries of the AP's authentication log.

However, if we allow APs to access the GM (or some third party) in their setup, we can improve the second scheme so that it satisfies the public detectability property. In an AP's setup of the improved scheme, not each AP but the GM (or the third party) generates the signatures $\{S_w\}$. Then no colluding subset of an AP and a user is able to execute the above attack, and therefore the improved scheme satisfies the public detectability property.

3.4 Selection of f_x^κ

The deterministic function f_x^κ plays central role in our protocol, and the choice of this functions influence the efficiency of our protocol. In this subsection, we discuss why we chose it to be $f_x^\kappa(X, w) = {g_X}^w {h_X}^{1/(x+w)}$, where g_X and h_X are deterministically computed from the value $\mathsf{Hash}(X)$.

The function takes two inputs, which is the identifier of AP, X, and the value w specifying that this is w-th authentication for the user. We represent the family of the set of the two inputs as $\{\mathcal{X}_\kappa\}$ and $\{\mathcal{Y}_\kappa\}$.

If we chose f_x^κ to be pseudorandom, namely its output is indistinguishable from random function then it would be sufficient to make our scheme secure. However, in our protocol we further need to prove knowledge of input to some output of function f_x^κ. We could not build an efficient proof if we choose f_x^κ to be one of the pseudorandom functions that we know of [BCK03,DN02,GGM86,NR97].

Instead, we introduce a non-pseudorandom function but one which we can construct an efficient proof, and one which we can prove the scheme to be secure. This property of this function can be generalized as to be called *partial* pseudorandom function family. That is, the function may not be pseudorandom, but if we restrict the domain of one of the input to be polynomial in regard to security parameter κ, the resulting function is pseudorandom. Since we only consider polynomial adversary, we can show that considering partial psuedorandom function is sufficient for the security of the schehe. The details of the proof is provided in the full paper.

Definition 2 Let $\{\mathcal{X}_\kappa\}$, $\{\mathcal{Y}_\kappa\}$, and $\{\mathcal{Z}_\kappa\}$ be families of sets, and $\{f_x^\kappa\}_\kappa$ be a function family of $f_x^\kappa : \mathcal{X}_\kappa \times \mathcal{Y}_\kappa \to \mathcal{Z}_\kappa$. We call the function family $\{f_x^\kappa\}_\kappa$ a *(secure) partial pseudorandom function family* if the following property is satisfied: for any polynomial $p(\kappa)$, and for any family $\{\mathcal{C}_\kappa\}$ of sets satisfying $\mathcal{C}_\kappa \subset \mathcal{Y}_\kappa$ and $\#\mathcal{C}_\kappa \leq p(\kappa)$, the family of restricted mappings $\{f_x^\kappa|_{\mathcal{X}_\kappa \times \mathcal{C}_\kappa}\}_\kappa$ is a secure pseudorandom function family.

The property of partial pseudorandomness is helpful in proving the property of the total anonymity. That is, if f_x^κ satisfies the partial pseudorandomness, the tag $(\tau, \hat{\tau}) = (f_x^\kappa(X, w), \mathsf{mik}^\ell f_x^\kappa(X, -w))$ is equivalent to some random pair. This means that $(\tau, \hat{\tau})$ do not reveal who is authenticated.

We next discuss why we set (g_X, h_X) to a hash value of some data in our construction. If we arbitrary chose g_X and h_X, an adversary $\mathcal{A}$ can choose $((g_X, h_X), (g_{X'}, h_{X'}))$ satisfying some polynomial time checkable relation, such as $(g_X, h_X) = ({g_{X'}}^2, {h_{X'}}^2)$. Then the polynomial time checkable relation, such as $f_x^\kappa(X, w) = f_x^\kappa(X', w')^2$, is maintained. This means that f_x^κ does not satisfy the partial pseudorandomness. Restricting (g_X, h_X) to be generated from hash function prevents this kind of attack. Proofs regarding that the proposed function satisfies partial pseudorandomness is provided in the full paper.

We note another important property of the function is collision resistance, which is defined below.

Definition 3 We say that $\{f_x^\kappa(X,w)\}_\kappa$ satisfies *collision resistance*, if it satisfies the following: for all X, a mapping $(x,w) \mapsto f_x^\kappa(X,w)$ is collision resistant.

The collision resistant property is helpful to prove the exculpability for users or the GM. If we use not f_x^κ but another function F_x^κ which does not satisfy the collision resistant property, then an adversary $\mathcal{A}$ can find two pairs (x_1,w_1) and (x_2,w_2) satisfying $F_{x_1}(X,w_1) = F_{x_2}(X,w_2)$. Let mik_i be the member identification key corresponding to x_i, and $(\tau_i,\hat{\tau}_i)$ be a tag $(f_{x_i}^\kappa(X,w_i), {\mathsf{mik}_i}^{\ell_i} f_{x_i}^\kappa(X,-w_i))$. Then two tags $(\tau_1,\hat{\tau}_1)$ and $(\tau_2,\hat{\tau}_2)$ satisfy $\tau_1 = F_{x_1}(X,w_1) = F_{x_2}(X,w_2) = \tau_2$. However, $(\tau_1/\tau_2)^{1/(\ell_1-\ell_2)}$ corresponds to neither mik_1 nor mik_2. Therefore, the tracing algorithm using (τ_1,τ_1') and (τ_2,τ_2') outputs some other user's ID or GM. This means that the k-TAA scheme does not satisfy exculpability for users or the GM.

We provide in the full paper that with this choice of the function the proposed scheme is secure.

3.5 Computational Costs of an Authentication

We show that the computational cost of an authentication of the first and the second schemes are $O(\log k)$ and $O(1)$ respectively. All we must show are that the condition (T1), (T2), and (T3') can be proved only with $O(1)$ exponentiations and that (T3) can be proved only with $O(\log k)$ exponentiations. See the full paper for the details of the validity proof.

The condition (T1) and (T3') can be proved only with $O(1)$ exponentiations, since this condition does not contain k. The condition (T2) is equal to the condition $(\tau,\hat{\tau}) = ({g_{v||k}}^w {h_{v||k}}^{1/(x+w)},\ {g_{v||k}}^w {h_{v||k}}^{1/(x-w)})$. Since $(g_{v||k}, h_{v||k}) = \mathsf{Hash}(v||k)$ is public information, one can prove the condition (T2) with $O(1)$ exponentiations too.

The condition (T3) is able to be proved using commitments $K(w)$, $\{K(w_i)\}$, and $K(u_i)$. Here $K(w)$, $K(w_i)$, and $K(u_i)$ are Pedersen commitments [P91] of w, i-th bit w_i of w, and i-th bit u_i of $k-w$. More precisely, one can prove the condition (T3) by proving the knowledge of $(\{w_i\},\{u_i\})$ satisfying the following conditions: $K(w) = \Pi_j K(w_j)^{2^j}$, $K(w)\Pi_j K(u_i)^{2^j} = K(k)$, $w_i, u_i \in \{0,1\}$ for $i = 0,\ldots,\log_2 k$. Here $K(k)$ is a commitment of k. From the above discussion, (T3) can be proved only with $O(\log k)$ exponentiations.

4 Security

We can show that our schemes are secure based on the following assumptions:

Definition 4 *(*Strong Diffie-Hellman (SDH) assumption *[BB04] on $(\mathcal{G},\mathcal{H},\mathcal{T})$) Let ϕ be the distorsion map from $\mathcal{H}$ to $\mathcal{G}$. Let $n = n(\kappa)$ be a polynomial and $\mathcal{A}$ be an adversary. Then $\mathsf{Prob}_\mathcal{A}(\kappa,n(\kappa)) = \Pr(v \leftarrow_R \mathcal{H}, u \leftarrow \phi(v), x \leftarrow_R \mathbb{Z}_q, \mathcal{A}(u,v,v^x \ldots, v^{x^n}) = (u^{1/(x+\beta)},\beta))$ is negligible for all $n(\kappa)$ and $\mathcal{A}$.*

Definition 5 *(*Decision Diffie-Hellman Inversion (DDHI) assumption *[BB04] on $\mathcal{U}$) Let $n = n(\kappa)$ be a polynomial and $\mathcal{A}$ be an adversary. For $b = 0,1$, we set $\mathsf{Prob}^b_\mathcal{A}(\kappa,n(\kappa)) = \Pr(g \leftarrow_R \mathcal{U}, x \leftarrow_R \mathbb{Z}_q, \mathcal{A}(h_b, g, g^{x^1},\ldots,g^{x^n}) = 1)$. Here h_0 is a randomly selected element of $\mathcal{U}$ and h_1 is the element $g^{1/x}$. Then for all $n(\kappa)$ and $\mathcal{A}$, $\mathsf{Adv}_\mathcal{A}(\kappa,n(\kappa)) = |\mathsf{Prob}^1_\mathcal{A}(\kappa,n(\kappa)) - \mathsf{Prob}^0_\mathcal{A}(\kappa,n(\kappa))|$ is negligible for κ.*

Our scheme satisfies the following:

Theorem 6. *Suppose that* $\{f_x^\kappa(X,w)\}_\kappa$ *is a secure collision resistant partial pseudorandom function family. Suppose also the SDH assumption [BB04] on* $(\mathcal{G},\mathcal{H},\mathcal{T})$ *and the random oracle assumption. Then the first scheme is secure and satisfies the public detectability. Suppose also that a signature scheme* $(\mathsf{SGen},\mathsf{Sig},\mathsf{SVer})$ *is existentially unforgeable even if an adversary knows signatures* $S_1,\ldots,S_{k(\kappa)}$ *on known messages* $1,\ldots,k(\kappa)$ *for all polynomial* $k(\kappa)$*. Then the second scheme is secure.*

From [BB04], the signature scheme $(\mathsf{SGen},\mathsf{Sig},\mathsf{SVer})$ satisfies the above condition under the SDH assumption. We next consider the security of $\{f_x^\kappa(X,w)\}_\kappa$.

Proposition 7 *Under the random oracle assumption and the DDHI assumption [BB04] on* $\mathcal{U}$*, the function family* $\{f_x^\kappa(X,w)\}_\kappa$ *is a secure collision resistant partial pseudorandom function family.*

From the above discussion, we can conclude the following theorem:

Theorem 8. *Under the SDH assumption on* $(\mathcal{G},\mathcal{H},\mathcal{T})$ *and the DDHI assumptions on* $\mathcal{U}$*, our two proposed schemes are secure in the random oracle. Moreover, under the same assumptions and in the same model, the first scheme satisfies the public detectability property.*

4.1 Sketch of the Security Proof

We sketch the proof of Theorem 6. See the full paper for the detailed proof. We first examine the security of the first scheme.

Total Anonymity: Let (x_u, r_u) be the member secret key of the target user u. Subinformation about x_u which an adversary can obtain are the following: (1) $A'_u = {a_1}^{x_u}{a_2}^{r'_u}$ and $\mathsf{mik}_u = f_{x_u}^\kappa(0,0)$, generated in the user u's joining protocol, and (2) $(\tau,\hat{\tau}) = (f_{x_u}^\kappa(v||k,w), {\mathsf{mik}_u}^\ell \cdot f_{x_u}^\kappa(v||k,-w))$ generated in each authentication. Since r'_u is randomly selected, A'_u gives no information about x_u.

From Proposition 7, $f_{x_u}^\kappa$ satisfies the partial pseudorandomness. Since the adversary is polynomial time algorithm, the oracles computes $f_{x_u}^\kappa$ only a polynomial number of times. Therefore, $\mathcal{A}$ cannot distinguish mik_u and $(\tau,\hat{\tau})$ from random elements. Hence $\mathcal{A}$ cannot distinguish which user is authenticated.

Exculpability for Users: In order to be authenticated on the behalf of the target user, $\mathcal{A}$ has to obtain the target user's secret key x_u. Subinformation about x_u which an adversary can obtain is the (1) and (2) described in the security discussion of the total anonymity. As in the case of the total anonymity, no adversary is able to obtain the target user's secret key x_u. Therefore, no adversary is able to be authenticated on the behalf of the target user.

Exculpability for the GM: This followed from the *coalition resistance* property [ACJT00,NN04] of the Furukawa-Imai scheme. Here the coalition resistance is the following property: an adversary, not colluding with the GM, cannot obtain a member public key/private key pair not generated in the joining protocols with the GM. It is well known that a secure group signature scheme satisfies the coalition resistance property [BMW03,BSZ05]. Hence, the Furukawa-Imai scheme (and therefore our first scheme) satisfies this property.

Suppose that an adversary of the exculpability property for the GM wins. In other words, suppose that there exist $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ and $(\tau', \hat{\tau}', \ell', \mathsf{pf}')$ in the authentication log of an AP such that $\tau = \tau'$ is satisfied and $(\hat{\tau}/\hat{\tau}')^{1/(\ell-\ell')}$ is not in the List. Since $\tau = \tau'$ is satisfied, and since f^{κ} is collision resistance, member public key/private key pairs used to compute τ and τ' are the same. Let $((\mathsf{mck}, \mathsf{mik}), \mathsf{msk})$ be this key pair. From the definition of $\hat{\tau}$ and $\hat{\tau}'$, $\mathsf{mik} = (\hat{\tau}/\hat{\tau}')^{1/(\ell-\ell')}$ is satisfied. From the coalition resistance property, this key was generated in a joining protocol with the GM. Therefore, mik is in List. It contradicts to the fact that $\mathsf{mik} = (\hat{\tau}/\hat{\tau}')^{1/(\ell-\ell')}$ is not in the List.

Exculpability for APs: This is clearly satisfied.

Public Detectability: Let N be the number of times an adversary $\mathcal{A}$ executes the joining with the GM. From the definition of the joining protocol of our first scheme, N is not more than the number of elements of List. Let k be the allowable number of an AP v colluding with $\mathcal{A}$. Suppose that $\mathcal{A}$ succeeds to output Log which contains more than kN elements. In each entry $(\tau, \hat{\tau}, \ell, \mathsf{pf})$ of Log, $(\tau, \hat{\tau}) = (f_x^{\kappa}(v||k, w), \mathsf{mik}^{\ell} \cdot f_x^{\kappa}(v||k, -w))$ has to be satisfied for some x and w, since $\mathcal{A}$ has generated the validity proof pf of $(\tau, \hat{\tau})$.

Since the coalition resistance property is satisfied, x is a part of a secret key generated in a joining with the GM. Hence, there are only N choices of x. Since $\mathcal{A}$ has proved the condition (T3) of 3.2, $1 \leq w \leq k$ is satisfied. Hence, there are only k choices of w. (We note that $1 \leq w \leq k$ has to be clearly satisfied even if $\mathcal{A}$ colludes with the AP). Therefore, the number of choices of x and w are respectively less than N and k. Therefore, an adversary generates at most kN elements $(\tau, \hat{\tau})$. This means that Log cannot have more than kN elements.

We now intuitively show the security of the second scheme. We only show that our second scheme satisfies the detectability property, since the proofs of the other requirements are similar to those of the first scheme.

Detectability: Let N be the number of times an adversary $\mathcal{A}$ executes the joining with the GM. In each authentication, $\mathcal{A}$ computes $(\tau, \hat{\tau}) = (f_x^{\kappa}(v||k, w), \mathsf{mik}^{\ell} \cdot f_x^{\kappa}(v||k, -w))$. Since $\mathcal{A}$ proves the condition (T3') of 3.3, $\mathcal{A}$ knows a signature S on w by the AP. Since the AP has published the signatures only on $1, \ldots, k$, and since $\mathcal{A}$ is not allowed to collude with the AP, $1 \leq w \leq k$ has to be satisfied. Hence, there are only k choices of w. Moreover, as in the case of the security discussion of the public detectability property of the first scheme, there are only N choices of x. Therefore, an adversary generates at most kN elements $(\tau, \hat{\tau})$. It means that Log cannot have more than kN elements.

5 Conclusion

We proposed two k-TAA schemes, one where the numbers of exponentiations in an authentication are $O(\log k)$ and the other $O(1)$. The proposed schemes are secure under the SDH assumption [BB04], the DDHI assumption [BB04], and the random oracle assumption.

We also proposed and formalized the *public detectability* requirement, and showed that the first scheme satisfies this requirement. The public detectability requires that anyone can verify that the AP indeed provided a fair limit to all users regarding the number of accesses to their service.

Acknowledgement

The authors would like to thank anonymous referees for their valuable comments and their help in improving the presentation of our paper.

References

[ACJT00] Ateniese, Camenisch, Joye, Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. CRYPTO 2000, pp. 255-270.

[AF96] Abe, Fujisaki, How to Date Blind Signatures, In ASIACRYPT'96, pp. 244-251.

[AM03] Ateniese Medeiros. Efficient Group Signatures without Trapdoors. ASIACRYPT'03, pp. 246-268.

[BCK03] Bellare, Canetti, Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. FOCS'96, 514-523.

[BMW03] Bellare, Micciancio, Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions In EUROCRYPT'03, pp. 614-629.

[BSZ05] Bellare, Shi Zhang. Foundations of Group Signatures: The Case of Dynamic Groups. CT-RSA'05, pp. 136-153.

[BB04] Boneh, Boyen. Short Signatures Without Random Oracles. EUROCRYPT'04, pp. 56-73.

[BBS04] Boneh, Boyen, Shacham. Short Group Signatures. CRYPTO'04, pp. 41-55.

[B00] Boudot. Efficient Proofs that a Committed Number Lies in an Interval. EUROCRYPT'00, pp 431-444.

[B93] Brands. An Efficient Off-line Electronic Cash System Based On The Representation Problem. TR. CS-R9323, Centrum voor Wiskunde en Informatica.

[BCC04] Brickell, Camenisch, Chen. Direct Anonymous Attestation. ACM-CCS'04, pp. 132-145.

[CDS94] Cramer, Damgård, Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. CRYPTO'94, pp. 174-187.

[CFT98] Chan, Frankel, Tsiounis. Easy Come - Easy Go Divisible Cash. EUROCRYPT '98, pp. 614-629.

[CG04] Camenisch, Groth. Group Signatures: Better Efficiency and New Theoretical Aspects. SCN'04, pp. 120-133.

[CHL05] J. Camenisch, S. Hohenberger and A. Lysyanskaya. Compact E-Cash. EUROCRYPT'05, pp. 302-321.

[CL02] Camenisch, Lysyanskaya. A Signature Scheme with Efficient Protocols. SCN'02, pp. 268-289.

[CL04] Camenisch, Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. CRYPTO'04, pp. 56-72.
[CH91] D. Chaum, E. van Heijst. Group signatures. EUROCRYPT'91, pp. 257-265.
[CP92] Chaum,Pedersen.Transferred Cash Grows in Size,EUROCRYPT'92,pp.390-407
[CT03] Canard, Traoré. List Signature Schemes and Application to Electronic Voting. International Workshop on Coding and Cryptography 2003, pp.24-28.
[CT04] Canard, Schoenmakers, Stam, Traoré. List Signature Schemes. Special Issue of the Journal Discrete Applied mathematics, 2005.
[DJ01] Damgård Jurik. A Generalization, a Simplification and Some Applications of Paillier's Probabilistic Public-key system. PKC'01. pp. 119-136.
[DN02] Damgård, Nielsen. Expanding Pseudorandom Functions; or: From Known-Plaintext Security to Chosen-Plaintext Security. CRYPTO'02, pp.449-464.
[DY05] Dodis, Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. PKC'05, pp.416-431
[FI05] Furukawa, Imai. An Efficient Group Signature Scheme from Bilinear Maps. ACISP'05, pp.455-467.
[FS01] Furukawa, Sako. An Efficient Scheme for Proving a Shuffle. CRYPTO'01,pp. 368-387.
[GGM86] Goldreich, Goldwasser, Micali. How to construct random function. J.ACM. 33(4): pp.797-807, 1986.
[KY04] Kiayias, Yung. Group Signatures: Provable Secure, Efficient Constructions and Anonymity from Trapdoor Holders. http://eprint.iacr.org/2004/076.ps
[KY05] A. Kiayias, M. Yung. Group Signatures with Efficient Concurrent Join. EUROCRYPT'05 pp. 198-214. Full version: http://eprint.iacr.org/2005/345.
[MSK02] Mitsunari, Sakai, Kasahara. A new traitor tracing. IEICE Trans. Vol. E85-A, No.2, pp.481-484, 2002.
[NHS99] Nakanishi, Haruna, Sugiyama. Unlinkable Electronic Coupon Protocol with Anonymity Control, ISW'99, pp. 37-46.
[NR97] Naor Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. FOCS'97. pp.458-467.
[NN04] Nguyen, Safavi-Naini. Efficient and Provably Secure Trapdoor-free Group Signature Schemes from Bilinear Pairings. ASIACRYPT'04, pp. 316-337.
[NN05] Nguyen, Safavi-Naini. Dynamic k-Times Anonymous Authentication. ACNS'05, pp. 318-333.
[Neff01] Neff. A Verifiable Secret Shuffle and its Application to E-Voting, ACM-CCS'01 pp. 116-125.
[OO98] Okamoto, Ohta. One-Time Zero-Knowledge Authentications and Their Applications to Untraceable Electronic Cash. IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, vol E81-A, No. 1, pp. 2-10, 1998.
[OMAFO99] Ookubo, Miura, Abe, Fujioka, Okamoto. An improvement of a practical secret voting scheme. ISW'99, pp. 37-46.
[PBF99] Pavlovski, Boyd, Foo. Detachable Electronic Coins. ICICS'99, pp. 54-70.
[P91] Torben P. Pedersen. A Threshold Cryptosystem without a Trusted Party. EUROCRYPT'91, pp.522-526.
[S00] Sako. Restricted Anonymous Participation. SCIS'00, B12. (Japanese).
[SK94] Sako, Kilian. Secure Voting using Partially Compatible Homomorphisms. CRYPTO '94, pp. 411–424.
[TF03] Teranishi, Furukawa. Tag Signature. SCIS'03, 6C-2. (Japanese. Preliminary version of [TFS04]).
[TFS04] Teranishi, Furukawa, Sako. k-Times Anonymous Authentication. ASIACRYPT'04, pp. 308-322.