A DAA Scheme Requiring Less TPM Resources

Liqun Chen

Hewlett-Packard Laboratories
liqun.chen@hp.com

Abstract. Direct anonymous attestation (DAA) is a special digital signature primitive, which provides a balance between signer authentication and privacy. One of the most interesting properties that makes this primitive attractive in practice is its construction of signers. The signer role of DAA is split between two entities, a principal signer (a trusted platform module (TPM)) with limited computational capability and an assistant signer (a computer platform into which the TPM is embedded) with more computational power but less security tolerance. Our first contribution in this paper is a new DAA scheme that requires very few TPM resources. In fact the TPM has only to perform two exponentiations for the DAA Join algorithm and three exponentiations for the DAA Signing algorithm. We show that this new scheme has better performance than the existing DAA schemes and is provable secure based on the q-SDH problem and DDH problem under the random oracle model. Our second contribution is a modification of the DAA security model defined in [13] to cover the property of non-frameability.
Keywords: direct anonymous attestation, trusted platform module, bilinear map.

1 Introduction

Many types of digital signatures have been developed to achieve signer authentication as well as signer privacy. Generally speaking, there are three categories of signature primitives depending on which type of public keys is used for signature verification. Given a signature, if a verifier makes use of the signer's public key, like an ordinary signature scheme, that shows the signer's unique information and therefore this type of signatures does not provide signer privacy. If a verifier makes use of a set of public keys each binding to one potential signer, such as ring signatures, designated verifier signatures and concurrent signatures, signer privacy is held but the level of privacy is dependent on the size of the public key set. If a verifier makes use of a group public key, such as group signatures and Direct Anonymous Attestation (DAA), signer privacy is also held and the level of privacy is dependent on the size of the group. When the size of a group is very large, for example all PCs from a well-known manufacturer, the third category will be the most suitable solution.

The concept and a concrete scheme of DAA were first introduced by Brickell, Camenisch, and Chen [11] for remote anonymous authentication of a trusted computing platform. The first DAA scheme was adopted by the Trusted Computing Group (TCG), an industry standardization body that aims to develop and promote an open industry standard for trusted computing hardware and software building blocks. The DAA scheme was specified in the TCG TPM Specification Version 1.2 [38] that has recently been adopted by ISO/IEC as an international standard [28]. A historical perspective on the development of DAA was provided by the DAA authors in [12].

A DAA scheme involves a set of issuers, signers, and verifiers. An issuer is in charge of verifying the legitimation of signers and of issuing a DAA credential (also called a DAA membership credential) to each signer. A signer can prove the membership to a verifier by providing a DAA signature. The verifier can verify the membership credential from the signature but he cannot learn the identity of the signer. The following two unique properties make DAA attractive in practice.

The first one is that the signer role of DAA is split between two entities, a principal signer with limited computational and storage capability, e.g. a trusted platform module (TPM), and an assistant signer with more computational power but less security tolerance, e.g. an ordinary computer platform (namely the Host with the TPM embedded in). The TPM is the real signer and holds the secret signing key, whereas the host helps the TPM to compute the signature under the credential, but is not allowed to learn the secret signing key and to forge such a signature without the TPM involvement.

The second one is to provide different degrees of privacy. A DAA scheme can be seen as a special group signature scheme without the feature of opening the signer's identity from its signature by the issuer. Interactions in DAA signing and verification are anonymous, that means the verifier, the issuer or both of them colluded cannot discover the signer's identity from its DAA signature. Instead of full-traceability as held in group signatures [5], DAA has user-controlled-traceability, that we mean the DAA signer is able to control if a verifier enables to determine whether any two signatures have been produced by the same signer. Moreover, the signer and verifier may negotiate as to whether or not the verifier is able to link different signatures signed by the signer.

DAA has drawn a lot of attention from both cryptographic researchers and industry. Many researchers proposed to use DAA in different applications; for instance, Pashalidis and Mitchell showed how to use DAA in a single sign-on application [34], Balfe, Lakhani and Paterson utilized a DAA scheme to achieve peer-to-peer networks [3], and Leung and Mitchell use a DAA scheme to build an authentication scheme for mobile environment [31]. Researchers have worked on security analysis of DAA; for example [2, 30, 35, 37]. Researchers have also worked on performance, implementation and revocation of DAA, e.g. [15, 17].

The original DAA scheme [11] and another DAA scheme by Ge and Tate [27] are based on the strong-RSA problem. We call them RSA-DAA for short. In an independent work [18], Canard and Traore proposed the concept of list signatures, in which signatures within a certain time frame are linkable. This property is similar to the user-controlled-traceability in DAA. Also in [18], the authors proposed a concrete list signature scheme, that, as the same as the first DAA scheme, is based on the strong RSA problem.

Recently, many researchers have been working on how to create DAA schemes with elliptic curves and pairings. We call these DAA schemes ECC-DAA for short. Generally speaking, ECC-DAA is more efficient in both computation cost and communication cost than RSA-DAA. Two of the most significant benefits are that the TPM's operation is much simpler and the key/signature length is much shorter in ECC-DAA than in RSA-DAA.

To our best knowledge, there are five ECC-DAA schemes available. The first one was proposed by Brickell, Chen and Li [13, 14]. This scheme is based on symmetric pairings. For the purpose of increasing implementation flexibility and efficiency, Chen, Morrissey and Smart proposed two extensions of this scheme [21–23]. Their schemes are based on asymmetric pairings. A flaw in the first one was pointed out by Li and further discussed in [20, 23]. Security of these three DAA schemes are based on the LRSW problem [33] and DDH problem. The other two DAA schemes were proposed by Chen and Feng [24] and Brickell and Li [16], respectively. Security of these two schemes are based on the q-SDH problem [8] and DDH problem.

We have two contributions in this paper. The first one is a new ECC-DAA scheme, which takes the advantages of multiple existing pairing-based DAA schemes. More specifically, our new scheme is a modification of the last two DAA schemes above. One of the most significant benefits is that the scheme requires very few TPM resources because it places a small computational requirement on a TPM. In fact the TPM has only to perform two exponentiations for the DAA Join protocol and three exponentiations for the DAA Signing protocol. This computational workload is equivalent to a couple of ordinary standard digital signatures, such as EC-DSA or EC-SDSA [29].

We will give some comparison between the proposed scheme and all the existing ECC-DAA schemes, and show that this new scheme has better performance than all the existing schemes. We will also provide a formal security proof of the scheme, based on the q-SDH problem and DDH problem under the random oracle model. The second contribution is a modification of the DAA security model defined in [13] to cover the property of non-frameability.

The rest of this paper is organized as follows. We first introduce some preliminaries in the next section, including the formal definition and security model of DAA, and definitions of pairings and relevant hard problems. We then specify our new DAA scheme in Section 3 and the corresponding security proofs in Section 4. We show some comparison between this scheme and all the existing ECC-DAA schemes in Section 5 that demonstrates the proposed scheme is the most efficient DAA scheme so far. We conclude the paper in Section 6.

2 Preliminaries

Throughout the paper, we will use some standard notation as follows. If S is any set then we denote the action of sampling an element from S uniformly at random and assigning the result to the variable x as $x \leftarrow S$. If A is any algorithm then we denote the action of obtaining x by running A on inputs $y_1, \ldots, y_n$ as $x \leftarrow A(y_1, \ldots, y_n)$. We denote concatenation of two date strings x and y as $x \| y$. We write $\{0,1\}^\ell$ for the set of binary strings of length ℓ and $\{0,1\}^*$ for the set of binary strings of arbitrary length.

2.1 Formal definition and security model of DAA

We recall the formal definition of DAA and modify the DAA security model described in [13] by adding the property of non-frameability (as described in [7]) or called exculpability (as described in [1]), i.e., the dishonest issuer and signers together are unable to create a judge-accepted proof that an honest signer produced a certain valid signature σ_0, e.g. it can be linked to some given signature σ_1 signed by the honest signer, unless this honest signer really did produce this signature σ_0.

A DAA scheme involves four types of players: a set of DAA issuers $\mathfrak{i}_k \in \mathfrak{I}$, TPM $\mathfrak{m}_i \in \mathfrak{M}$, host $\mathfrak{h}_i \in \mathfrak{H}$ and verifier $\mathfrak{v}_j \in \mathfrak{V}$. The index values, k, i, j, are polynomial. $\mathfrak{m}_i$ and $\mathfrak{h}_i$ form a computer platform in the trusted computing environment and share the role of a DAA signer. The following three cases are considered in the security model: (1) neither $\mathfrak{m}_i$ nor $\mathfrak{h}_i$ is corrupted by an adversary, (2) both of them are corrupted, and (3) $\mathfrak{h}_i$ is corrupted but not $\mathfrak{m}_i$. Like in other DAA papers, we do not consider the case that $\mathfrak{m}_i$ is corrupted but not $\mathfrak{h}_i$, because $\mathfrak{m}_i$ plays a principal role of the signer, i.e. holding the private signing key. Throughout the paper, for the purpose of simplicity, we may omit some of the index values if it does not occur any confusion; for example, we make use of $\mathfrak{i}$ instead of $\mathfrak{i}_k$.

A DAA scheme $\mathcal{DAA} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link})$ consists of the following five polynomial-time algorithms and protocols:

- Setup: On input of a security parameter 1^t, $\mathfrak{i}$ uses this randomized algorithm to produce a pair $(\mathsf{isk}, \mathsf{par})$, where isk is the issuer's secret key, and par is the global public parameters for the system, including the issuer's public key ipk, a description of a DAA credential space C, a description of a finite message space M and a description of a finite signature space Σ. We will assume that par are publicly known so that we do not need to explicitly provide them as input to other algorithms.
- Join: This protocol, run between a signer $(\mathfrak{m}_i, \mathfrak{h}_i)$ and an issuer $\mathfrak{i}$, consists of two randomized algorithms, namely $\mathsf{Join_t}$ and $\mathsf{Join_i}$. $\mathfrak{m}_i$ uses $\mathsf{Join_t}$ to produce a pair $(\mathtt{tsk}_i, \mathsf{comm}_i)$, where $\mathtt{tsk}_i$ is the TPM's secret key and comm_i is a commitment of $\mathtt{tsk}_i$. On input of comm_i and isk, $\mathfrak{i}$ uses $\mathsf{Join_i}$ to produce cre_i, which is a DAA credential associated with $\mathtt{tsk}_i$. The value cre_i is given to both $\mathfrak{m}_i$ and $\mathfrak{h}_i$, but the value $\mathtt{tsk}_i$ is known to $\mathfrak{m}_i$ only.
- Sign: On input of $\mathtt{tsk}_i$, cre_i, a basename bsn_j (the name string of $\mathfrak{v}_j$ or a special symbol $\perp$), and a message m that includes the data to be signed and the verifier's nonce n_V for freshness, $\mathfrak{m}_i$ and $\mathfrak{h}_i$ run this protocol to produce a randomized signature σ on m under $(\mathtt{tsk}_i, \mathsf{cre}_i)$ associated with bsn_j. The basename bsn_j is used for controlling the linkability.

– Verify: On input of m, bsn_j, a candidate signature σ for m, and a set of rogue signers' secret keys $\mathsf{RogueList}$, $\mathfrak{v}_j$ uses this deterministic algorithm to return either 1 (accept) or 0 (reject). Note that how to build the set of $\mathsf{RogueList}$ is out the scope of the DAA scheme.
– Link: On input of two signatures σ_0 and σ_1, $\mathfrak{v}_j$ uses this deterministic algorithm to return 1 (linked), 0 (unlinked) or $\perp$ (invalid signatures). Link will output $\perp$ if, by using an empty $\mathsf{RogueList}$ (which means to ignore the rogue TPM check), either $\mathsf{Verify}(\sigma_0) = 0$ or $\mathsf{Verify}(\sigma_1) = 0$ holds. Otherwise, Link will output 1 if signatures can be linked or 0 if the signatures cannot be linked. Note that, unlike Verify, the result of Link is not relied on whether the corresponding $\mathtt{tsk} \in \mathsf{RogueList}$ or not.

In this security model, a DAA scheme must hold the notions of *correctness*, *user-controlled-anonymity* and *user-controlled-traceability*. They are defined as follows.

Correctness If both the signer and verifier are honest, that implies $\mathtt{tsk}_i \notin \mathsf{RogueList}$, the signatures and their links generated by the signer will be accepted by the verifier with overwhelming probability. This means that the above DAA algorithms must meet the following consistency requirement. If

$$(\mathsf{isk}, \mathsf{par}) \leftarrow \mathsf{Setup}(1^t),$$
$$(\mathtt{tsk}_i, \mathsf{cre}_i) \leftarrow \mathsf{Join}(\mathsf{isk}, \mathsf{par}), \text{ and}$$
$$(m_b, \sigma_b) \leftarrow \mathsf{Sign}(m_b, \mathsf{bsn}_j, \mathtt{tsk}_i, \mathsf{cre}_i, \mathsf{par})|_{b=\{0,1\}},$$

then we must have

$$1 \leftarrow \mathsf{Verify}(m_b, \mathsf{bsn}_j, \sigma_b, \mathsf{par}, \mathsf{RogueList})|_{b=\{0,1\}} \text{ and}$$
$$1 \leftarrow \mathsf{Link}(\sigma_0, \sigma_1, \mathsf{par})|_{\mathsf{bsn}_j \neq \perp}.$$

User-Controlled-Anonymity The notion of user-controlled-anonymity is defined via a game played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ as follows:

– *Initial:* $\mathcal{C}$ runs $\mathsf{Setup}(1^t)$ and gives the resulting isk and par to $\mathcal{A}$. Alternatively, $\mathcal{C}$ receives par from $\mathcal{A}$ with a request for initiating the game, and then verifies the validation of the par by checking whether each element of the par is in the right groups or not.
– *Phase 1:* $\mathcal{C}$ is probed by $\mathcal{A}$ who makes the following queries:
  • Sign. $\mathcal{A}$ submits a signer's identity ID, a basename bsn (either $\perp$ or a data string) and a message m of his choice to $\mathcal{C}$, who runs Sign to get a signature σ and responds with σ.
  • Join. $\mathcal{A}$ submits a signer's identity ID of his choice to $\mathcal{C}$, who runs $\mathsf{Join}_{\mathsf{t}}$ with $\mathcal{A}$ to create $\mathtt{tsk}$ and to obtain cre from $\mathcal{A}$. $\mathcal{C}$ verifies the validation of cre and keeps $\mathtt{tsk}$ secret.
  • Corrupt. $\mathcal{A}$ submits a signer's identity ID of his choice to $\mathcal{C}$, who responds with the value $\mathtt{tsk}$ of the signer.

– *Challenge:* At the end of Phase 1, $\mathcal{A}$ chooses two signers' identities ID_0 and ID_1, a message m and a basename bsn of his choice to $\mathcal{C}$. $\mathcal{A}$ must not have made any Corrupt query on either ID_0 or ID_1, and not have made the Sign query with the same bsn if $\mathsf{bsn} \neq \perp$ with either ID_0 or ID_1. To make the challenge, $\mathcal{C}$ chooses a bit b uniformly at random, signs m associated with bsn under $(\mathtt{tsk}_b, \mathsf{cre}_b)$ to get a signature σ and returns σ to $\mathcal{A}$.
– *Phase 2:* $\mathcal{A}$ continues to probe $\mathcal{C}$ with the same type of queries that it made in Phase 1. Again, it is not allowed to corrupt any signer with the identity either ID_0 or ID_1, and not allowed to make any Sign query with bsn if $\mathsf{bsn} \neq \perp$ with either ID_0 or ID_1.
– *Response:* $\mathcal{A}$ returns a bit b'. We say that the adversary wins the game if $b = b'$.

Definition 1. *Let $\mathcal{A}$ denote an adversary that plays the game above. We denote by $\mathbf{Adv}[\mathcal{A}^{anon}_{\mathcal{DAA}}] = |\mathbf{Pr}[b' = b] - 1/2|$ the advantage of $\mathcal{A}$ in breaking the user-controlled-anonymity of $\mathcal{DAA}$. We say that a DAA scheme is user-controlled-anonymous if for any probabilistic polynomial-time adversary $\mathcal{A}$, the quantity $\mathbf{Adv}[\mathcal{A}^{anon}_{\mathcal{DAA}}]$ is negligible.*

Note that a value is *negligible* means this value is a function $\epsilon(t)$, which is said to be *negligible* in the parameter t if $\forall\ c \geq \mathbb{Z}_{>0}\ \exists\ t_c \in \mathbb{R}_{>0}$ such that $\forall\ t > t_c, \epsilon(t) < t^{-c}$.

User-Controlled-Traceability The notion of User-Controlled-Traceability is defined via a game played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ as follows:

– *Initial:* There are two initial cases. In Initial Case 1. $\mathcal{C}$ executes $\mathsf{Setup}(1^t)$ and gives the resulting par to $\mathcal{A}$, and $\mathcal{C}$ keeps isk secret. In Initial Case 2. $\mathcal{C}$ receives par from $\mathcal{A}$ and does not know the value of isk.
– *Probing:* $\mathcal{C}$ is probed by $\mathcal{A}$ who makes the following queries:
  • Sign. The same as in the game of user-controlled-anonymity.
  • Semi-sign. $\mathcal{A}$ submits a signer's identity ID along with the data transmitted from $\mathfrak{h}_i$ to $\mathfrak{m}_i$ in Sign of his choice to $\mathcal{C}$, who acts as $\mathfrak{m}_i$ in Sign and responds with the data transmitted from $\mathfrak{m}_i$ to $\mathfrak{h}_i$ in Sign.
  • Join. There are three join cases of this query; the first two are used associated with the Initial Case 1, and the last one is used associated with the Initial Case 2. Suppose that $\mathcal{A}$ does not use a single ID for more than one join case or more than one time.
    ∗ Join Case 1: $\mathcal{A}$ submits a signer's identity ID of his choice to $\mathcal{C}$, who runs Join to create $\mathtt{tsk}$ and cre for the signer, and finally $\mathcal{C}$ sends cre to $\mathcal{A}$ and keeps $\mathtt{tsk}$ secret.
    ∗ Join Case 2: $\mathcal{A}$ submits a signer's identity ID with a $\mathtt{tsk}$ value of his choice to $\mathcal{C}$, who runs Join_i to create cre for the signer and puts the given $\mathtt{tsk}$ into the list of $\mathsf{RogueList}$. $\mathcal{C}$ responds $\mathcal{A}$ with cre.
    ∗ Join Case 3: $\mathcal{A}$ submits a signer's identity ID of his choice to $\mathcal{C}$, who runs Join_t with $\mathcal{A}$ to create $\mathtt{tsk}$ and to obtain cre from $\mathcal{A}$. $\mathcal{C}$ verifies the validation of cre and keeps $\mathtt{tsk}$ secret.

- Corrupt. This is the same as in the game of user-controlled-anonymity, except that at the end $\mathcal{C}$ puts the revealed $\mathtt{tsk}$ into the list of $\mathsf{RogueList}$.

– *Forge:* $\mathcal{A}$ returns a signer's identity ID, a signature σ, its signed message m and the associated basename bsn. We say that the adversary wins the game if either of the following two situations is true:
  1. With the Initial Case 1 ($\mathcal{A}$ does not have access to isk),
     (a) $\mathsf{Verify}(m, \mathsf{bsn}, \sigma, \mathsf{RogueList}) = 1$ (accepted), but σ is neither a response of the existing Sign queries nor a response of the existing Semi-sign queries (partially); and/or
     (b) In the case of $\mathsf{bsn} \neq \perp$, there exists another signature σ' associated with the same identity and bsn, and the output of $\mathsf{Link}(\sigma, \sigma')$ is 0 (unlinked).
  2. With the Initial Case 2 ($\mathcal{A}$ knows isk), the same as the item (a), in the condition that the secret key $\mathtt{tsk}$ used to create σ was generated in the Join Case 3 (i.e., $\mathcal{A}$ does not have access to $\mathtt{tsk}$).

Definition 2. *Let $\mathcal{A}$ be an adversary that plays the game above. We denote $\mathbf{Adv}[\mathcal{A}^{trace}_{\mathcal{DAA}}] = \mathbf{Pr}[\mathcal{A}\ wins]$ as the advantage that $\mathcal{A}$ breaks the user-controlled-traceability of $\mathcal{DAA}$. We say that a DAA scheme is user-controlled-traceable if for any probabilistic polynomial-time adversary $\mathcal{A}$, the quantity $\mathbf{Adv}[\mathcal{A}^{trace}_{\mathcal{DAA}}]$ is negligible.*

Note that in the above game of the user-controlled-traceability, we allow the adversary to corrupt the issuer. This is an important difference from the game in [13], since it covers the requirement of non-frameability or called exculpabilty.

2.2 Pairings and relevant hard problems

Our new DAA scheme is based on asymmetric pairings. As discussed in [21], it will avoid the poor security level scaling problem in symmetric pairings and allow one to implement the DAA scheme efficiently at hight security levels. Throughout we let $\mathbb{G}_1 = \langle P \rangle, \mathbb{G}_2 = \langle Q \rangle$ and $\mathbb{G}_T$ be groups of large prime exponent $p \approx 2^t$ for security parameter t. All the three groups will be written multiplicatively. If $\mathbb{G}$ is some group then we use the notation $\mathbb{G}^\times$ to mean the non-identity elements of $\mathbb{G}$.

Definition 3 (Pairing). *A* pairing (or bilinear map) *is a map $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that:*

1. *The map $\hat{t}$ is* bilinear. *This means that $\forall P, P' \in \mathbb{G}_1$ and $\forall Q, Q' \in \mathbb{G}_2$ that*
   – $\hat{t}(P \cdot P', Q) = \hat{t}(P, Q) \cdot \hat{t}(P', Q) \in \mathbb{G}_T$.
   – $\hat{t}(P, Q \cdot Q') = \hat{t}(P, Q) \cdot \hat{t}(P, Q') \in \mathbb{G}_T$.
2. *The map $\hat{t}$ is* non-degenerate. *This means that*
   – $\forall P \in \mathbb{G}_1^\times \exists Q \in \mathbb{G}_2$ *such that* $\hat{t}(P, Q) \neq 1_{\mathbb{G}_T} \in \mathbb{G}_T$.
   – $\forall Q \in \mathbb{G}_2^\times \exists P \in \mathbb{G}_1$ *such that* $\hat{t}(P, Q) \neq 1_{\mathbb{G}_T} \in \mathbb{G}_T$.
3. *The map $\hat{t}$ is* computable *i.e. there exist some polynomial time algorithm to compute $\hat{t}(P, Q) \in \mathbb{G}_T$ for all $(P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2$.*

Our DAA scheme is based on the pairing based signature scheme that is used by Delerablee-Pointcheval [25] as a group membership certificate and also mentioned by Boneh-Boyen-Shacham in [9] as an alternative group member credential. This scheme (called the BBS signature scheme in the later part of this paper) is given by the following triple of algorithms:

- **KeyGeneration:** Select $P_1, P_2 \leftarrow \mathbb{G}_1$, $Q \leftarrow \mathbb{G}_2$ and $x \leftarrow \mathbb{Z}_p^*$, and compute $X \leftarrow Q^x \in \mathbb{G}_2$. The public key is the tuple (P_1, P_2, Q, X) and the private key is x.
- **Signing:** On input of a message $m \in \mathbb{Z}_p$ the signer chooses $e \leftarrow \mathbb{Z}_p$ and computes $A \leftarrow (P_1 \cdot P_2^m)^{1/(x+e)} \in \mathbb{G}_1$. The signature on m is $\sigma \leftarrow (A, e)$.
- **Verification:** To verify a signature σ on a message m the verifier checks whether $\hat{t}(A, X \cdot Q^e) = \hat{t}(P_1 \cdot P_2^m, Q)$.

The security of the above signature scheme is related to the hardness of the q-SDH problem introduced by Boneh and Boyen [8]. The q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows:

Definition 4 (q-SDH). *Given a $(q+2)$-tuple $(P, Q, Q^x, Q^{x^2}, ..., Q^{x^q}) \in \mathbb{G}_1 \times \mathbb{G}_2^{q+1}$ as input output a pair $(e, P^{1/(x+e)})$ where $e \in \mathbb{Z}_p^*$. An algorithm $\mathcal{A}$ has advantage ϵ in solving q-SDH in $(\mathbb{G}_1, \mathbb{G}_2)$ if*

$$P_r[\mathcal{A}(P, Q, Q^x, Q^{x^2}, ..., Q^{x^q}) = (e, P^{1/(x+e)})] \geq \epsilon,$$

where the probability is over the random choice of generator Q in $\mathbb{G}_2$ (with $P \leftarrow \psi(Q)$), of x in $\mathbb{Z}_p^*$ and of the random bits of $\mathcal{A}$. We say that the (q, t, ϵ)-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no t-time algorithm has advantage at least ϵ in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

As the same as the DAA scheme in [23], our DAA scheme requires the DDH problem for $\mathbb{G}_1$ to be hard. The formal definition of this problem is defined as follows:

Definition 5 ($\mathbb{G}_1$-DDH). *We define the $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{DDH}}(t)$ of an $\mathbb{G}_1$-DDH adversary $\mathcal{A}$ against the set of parameters $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q, p, \hat{t})$ as*

$$\begin{aligned}&\big|\Pr\left[x, y, z \leftarrow \mathbb{Z}_p; X \leftarrow xP, Y \leftarrow yP, Z \leftarrow zP; \mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{t}, P, Q, X, Y, Z, p) = 1\right]\\ &- \Pr\left[x, y \leftarrow \mathbb{Z}_p; X \leftarrow xP, Y \leftarrow yP; Z \leftarrow xyP; \mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{t}, P, Q, X, Y, Z, p) = 1\right]\big|\end{aligned}$$

We then say a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, Q, p, \hat{t})$ satisfies the DDH assumption for $\mathbb{G}_1$ if for any p.p.t. *adversary $\mathcal{A}$ its advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{DDH}}(t)$ is negligible in t.*

Often this problem in the context of pairing groups is called the *external Diffie–Hellman* problem, or the XDH problem.

3 The Proposed DAA Scheme

In this section, we give a detailed description of the new DAA scheme. Before proceeding we note a general point which needs to be born in mind for each of the following protocols and algorithms. Every group element received by any party needs to be checked that it lies in the correct group, and in particular does not lie in some larger group which contains the specified group. This is to avoid numerous attacks such as those related to small subgroups etc (e.g. [10, 32]). In asymmetric pairings this is particularly important since $\mathbb{G}_1$ and $\mathbb{G}_2$ can be considered as distinct subgroups of a large group $\mathbb{G}$. If transmitted elements are actually in $\mathbb{G}$, as opposed to $\mathbb{G}_1$ and $\mathbb{G}_2$, then various properties can be broken such as anonymity and linkability.

Hence, our security proofs implicitly assume that all transmitted group elements are indeed elements of the specified groups. For the situation under consideration, namely Type-III pairings [26], efficient methods for checking subgroup membership are given in [19]. Note, we do not count the cost of these subgroup checks in our performance considerations later on, as their relative costs can be quite dependent on the specific groups and security parameters under consideration.

3.1 The Setup Algorithm

On input of the security parameter 1^t, the setup algorithm executes the following:

1. Select three groups, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, of sufficiently large prime order p along with a pairing $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. Select four random generators, P_1, P_2, P_3 and Q, such that $\mathbb{G}_1 = \langle P_1 \rangle = \langle P_2 \rangle = \langle P_3 \rangle$ and $\mathbb{G}_2 = \langle Q \rangle$ and compute $T_1 = \hat{t}(P_1, Q)$, $T_2 = \hat{t}(P_2, Q)$ and $T_3 = \hat{t}(P_3, Q)$. Select five hash functions $H_1 : \{0,1\}^* \mapsto \mathbb{Z}_p$, $H_2 : \{0,1\}^* \mapsto \mathbb{Z}_p$, $H_3 : \{0,1\}^* \mapsto \mathbb{G}_1$, $H_4 : \{0,1\}^* \mapsto \mathbb{Z}_p$ and $H_5 : \{0,1\}^* \mapsto \mathbb{Z}_p$.
2. For each issuer $\mathfrak{i} \in \mathfrak{I}$, select an integer $x \leftarrow \mathbb{Z}_p$ and compute $X = Q^x \in \mathbb{G}_2$ and $T_4 = \hat{t}(P_3, X)$. The issuer secret key isk is assigned to be x and the corresponding public key ipk is assigned to be X.
3. For each TPM $\mathfrak{m} \in \mathfrak{M}$, select a sufficiently large integer $\mathsf{DAAseed}$ at random (e.g. choose $\mathsf{DAAseed}$ from $\{0,1\}^t$) and keep it inside of the TPM secretly.
4. Describe a DAA credential space C, a finite message space M and a finite signature space Σ. The spaces C and Σ will be defined in the Join protocol and Sign protocol respectively. The space M is dependent upon applications.
5. Finally, the system public parameters par are set to be ($\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, p, $\hat{t}$, P_1, P_2, P_3, Q, T_1, T_2, T_3, T_4, H_1, H_2, H_3, H_4, H_5, ipk) together with C, M and Σ, and are published.

The group order p is selected so that solving the $\mathbb{G}_1$-DDH problem or the q-SDH problem takes time 2^t. The three generators P_1, P_2, P_3 are selected so that the discrete logarithm relation between each other, e.g. $\log_{P_1} P_2$, is unknown. Including the four pairing values, T_1, T_2, T_3 and T_4, in the published par is optional; alternatively these values are computed by hosts and verifiers. To make sure these four values are formed correctly, it is recommended that each host and verifier should compute them once before storing them with other values of par.

We do not specify that issuers supply a proof of correctness of their public keys, i.e. that they know the underlying secret key value, or that a given user checks the correctness of the issuer public keys. Instead, during the join protocol, the correctness of issuer public keys are verified for any DAA credential issued by a given issuer. Also during the verification algorithm, the correctness of issuer public keys are verified for any signature produced from such a credential issued by a given issuer.

The value $\mathsf{DAAseed}$ is created by a given TPM and never leave the TPM. We assume that prior to any system setup each TPM has its private endorsement key SK embedded into it and that each issuer has access to the corresponding public endorsement key PK. This key pair is used to build up an authentic channel between the TPM and issuer in the following Join protocol.

3.2 The Join Protocol

This protocol is run between a given TPM $\mathfrak{m} \in \mathfrak{M}$, the corresponding Host $\mathfrak{h} \in \mathfrak{H}$ and an Issuer $\mathfrak{i} \in \mathfrak{I}$. We first give an overview of how a general Join protocol proceeds. There are 3 main stages to a Join protocol. First the TPM $\mathfrak{m}$ generates a DAA secret $f \in \mathbb{Z}_p$ using the value K_I provided by the issuer, an account number cnt provided by the host and its internal secret seed $\mathsf{DAAseed}$[1]. The TPM then computes a commitment on this value, i.e. $F = P_2^f \in \mathbb{G}_2$, along with a proof of possession of this value, i.e. (v, w), and passes the commitment and proof to its Host who adds these to a list of commitments and forwards them to the Issuer. In the second stage the issuer performs some checks on the commitment and proof it receives and, if these correctly verify, computes a credential, cre, and then sends it to the host. The final stage of a Join protocol is the Host verifies the correctness of the credential.

We note that one of the reasons why our DAA scheme is more efficient than the DAA schemes in [16, 24] is that we make use of a simplified construction of the DAA credential cre. In our scheme $\mathsf{cre} = (A, e)$ where $A = (P_1 \cdot P_2^f)^{1/(x+e)}$, but in the both schemes of [16, 24], $\mathsf{cre} = (A, e, y)$ where $A = (P_1 \cdot P_2^f \cdot P_3^y)^{1/(x+e)}$ and the value y is contributed by both the TPM and Issuer. Our security proof in Section 4 will show that our cre construction is sufficient for achieving required security features of the new scheme.

Our protocol proceeds as shown in Figure 1. The following notes should be kept in mind when examining this protocol.

[1] In this paper, we refer the value K_I as the identifier of the issuer $\mathfrak{i}$. Note that for a given issuer a TPM could compute many values of f, dependent on the values of K_I and cnt. The purpose of using K_I and cnt can be found in [11, 12].

TPM ($\mathfrak{m}$)		Host ($\mathfrak{h}$)		Issuer ($\mathfrak{i}$)
				$n_I \leftarrow \{0,1\}^t$
$\mathsf{str} \leftarrow X \| P_1 \| P_2 \| P_3 \| Q \| n_I$	$\xleftarrow{\text{req}}$		$\xleftarrow{\text{req}}$	$\text{req} \leftarrow n_I$
$f \leftarrow H_1(\mathsf{DAAseed} \| \mathsf{cnt} \| \mathrm{K}_I)$				
$u \leftarrow \mathbb{Z}_p$				
$F \leftarrow P_2^f ; U \leftarrow P_2^u$				
$v \leftarrow H_2(\mathsf{str} \| F \| U)$				
$w \leftarrow u + f \cdot v \pmod p$				
$\mathsf{comm} \leftarrow (F, v, w, n_I)$	$\xrightarrow{\mathsf{comm}}$		$\xrightarrow{\mathsf{comm}}$	If $n_I \notin \{\text{req}\}$, or
				$U' \leftarrow P_2^w \cdot F^{-v}$
				$\mathsf{str} \leftarrow X \| P_1 \| P_2 \| P_3 \| Q \| n_I$
				$v \neq H_2(\mathsf{str} \| F \| U')$
				then **abort**
				$e \leftarrow \mathbb{Z}_p$
				$A \leftarrow (P_1 \cdot F)^{1/(x+e)}$
		If $\hat{t}(A, X \cdot Q^e)$	$\xleftarrow{\mathsf{cre}}$	$\mathsf{cre} \leftarrow (A, e)$
		$\neq \hat{t}(P_1 \cdot F, Q)$		
		then **abort**		

Fig. 1. The Join Protocol

- We note that the nonce n_I on which the proof (v, w) is generated must be one that was sent out by the issuer. The host reminds the issuer of the value of n_I it was sent in $\mathsf{comm}_{\text{req}}$ and the issuer then checks this against its records.

- The triple (F, v, w) is essentially a proof of knowledge of the discrete logarithm of the value F. In contrast with the RSA-DAA schemes [11, 27] we do not require a relatively complicated proof of knowledge of the correctness of a given commitment. Instead, the proof of knowledge is provided by a very efficient Schnorr signature [36] on the value F computed using the secret key f. This is the same as in some existing pairing-based DAA schemes [13, 21, 23].

- The communication between the TPM and issuer is via the host. However, this communication must be authentic, that we mean the issuer must be sure that he only creates the DAA credential for a genuine TPM. An authentic channel between the TPM and issuer can be built by using the TPM endorsement key pair $(\mathsf{SK}, \mathsf{PK})$. We suggest using the same mechanism as in the RSA-DAA scheme [11, 38] to achieve this.

- The credential computed by the issuer $\mathfrak{i}$ is the BBS signature on the value f signed in a blind manner, that we mean the issuer $\mathfrak{i}$ does not known this value, although $\mathfrak{i}$ is convinced of the possession of f from verifying its commitment. Once a credential is issued from $\mathfrak{i}$, the Host $\mathfrak{h}_i$ verify that this credential is correctly formed.

3.3 The Sign Protocol

This protocol is run between a given TPM $\mathfrak{m} \in \mathfrak{M}$ and the corresponding Host $\mathfrak{h} \in \mathfrak{H}$. During the protocol $\mathfrak{m}$ and $\mathfrak{h}$ work together to produce a DAA signature on some message. The signature should prove knowledge of a discrete logarithm f, knowledge of a valid credential cre and that this credential was computed for the same value f by a given Issuer $\mathfrak{i} \in \mathfrak{I}$. We note that the Host will know a lot of the values needed in the computation and will be able to take on a lot of the computational workload. However, if the TPM has not had its secret f published (i.e. it is not a rogue module) then the Host $\mathfrak{h}$ will not know f and will be unable to compute the whole signature without the aid of the TPM. Therefore, we say that the TPM is the real signer and the Host is a helper.

TPM ($\mathfrak{m}$)		Host ($\mathfrak{h}$)
		$\mathsf{msg} \in \mathsf{M}$
		If $\mathsf{bsn} = \perp$ then $J \leftarrow \mathbb{G}_1$
		else $J \leftarrow H_3(\mathsf{bsn})$
		Either $n_V \leftarrow \{0,1\}^t$ or receive
		$n_V \in \{0,1\}^t$ from the verifier
		$\mathsf{str} \leftarrow X \Vert P_1 \Vert P_2 \Vert P_3 \Vert Q \Vert n_V$
		$a, r_a, r_e, r_{ae} \leftarrow \mathbb{Z}_p$
		$R \leftarrow A \cdot P_3^a$
		$\hat{S} \leftarrow T^{r_e} \cdot T_3^{a \cdot r_e + r_{ae}} \cdot T_4^{r_a}$
$r_f \leftarrow \mathbb{Z}_p$	$\xleftarrow{h,\ J,\ \hat{S},\ \mathsf{msg}}$	$h \leftarrow H_4(\mathsf{str} \Vert R)$
$K \leftarrow J^f; L \leftarrow J^{r_f}$		
$S \leftarrow \hat{S} \cdot T_2^{r_f}$		
$n_T \leftarrow \{0,1\}^t$		
$c \leftarrow H_5(h \Vert J \Vert K \Vert L \Vert S \Vert \mathsf{msg} \Vert n_T)$		
$s_f \leftarrow r_f + f \cdot c \pmod p$	$\xrightarrow{(K,\ c,\ s_f,\ n_T)}$	$s_a = r_a + a \cdot c \pmod p$
		$s_e = r_e - e \cdot c \pmod p$
		$s_{ae} = r_{ae} + a \cdot e \cdot c \pmod p$
		$\sigma \leftarrow (R, J, K, c, s_f, s_a, s_e, s_{ae})$

Fig. 2. The Sign Protocol

The protocol then proceeds as in Figure 2, so as to produce the signature σ. We note that in this scheme, the Host $\mathfrak{h}$ precomputes $T = \hat{t}(A, Q)$ and stores it as a long-term parameter. In Table 1 of Section 5, we do not list this pairing computation in the signing computational cost since it will only be computed once.

Again we provide some notes as to the rationale behind some of the steps:

– We let msg denote the message to be signed. As the same as in the original DAA scheme [11], msg is presented as $b||\mathsf{msg}'$ where $b = 0$ means that the message msg' is generated by the TPM and $b = 1$ means that msg' was input to the TPM by the Host. In the case that msg' is generated by the TPM,

for example, the TPM creates a cryptographic key, called an Attestation Identity Key (AIK), and forms its public key and relevant public system parameters as msg'. A DAA signature on an AIK is used as a self-certificate. For the purpose of this paper, we leave the specific details of how such signed messages are created by the TPM as an implementation detail. If msg' is not generated by the TPM, it may either be chosen/selected by the Host, or passed to the Host by the verifier. We let bsn denote the base name, that may either be chosen/selected by the Host, or passed to the Host by the verifier.
- During the run of the protocol two nonces are used: one from the verifier n_V and one from the TPM n_T. In most applications of the Sign protocol, the signature is generated as a request from the verifier, and the verifier supplies its own value of n_V, to protect against replays of previously requested signatures. If a signature is produced in an off-line manner we allow the Host to generate its own value of n_V. These are used to ensure each signature is different from previous signatures and to ensure no adversarially controlled TPM and Host pair, or no honest TPM and adversarially controlled Host, can predict or force the value of a given signature.
- We note that the correctness of the protocol holds because of the following equation:

$$
\begin{aligned}
S &= \hat{S} \cdot T_2^{r_f} \\
&= T^{r_e} \cdot T_3^{a \cdot r_e + r_{ae}} \cdot T_4^{r_a} \cdot T_2^{r_f} \\
&= \hat{t}(A,Q)^{r_e} \cdot T_3^{a \cdot r_e + r_{ae}} \cdot T_4^{r_a} \cdot T_2^{r_f} \\
&= (\hat{t}(A,Q) \cdot T_3^{a})^{r_e} \cdot T_3^{r_{ae}} \cdot T_4^{r_a} \cdot T_2^{r_f} \\
&= (\hat{t}(A,Q) \cdot \hat{t}(P_3,Q)^{a})^{r_e} \cdot T_3^{r_{ae}} \cdot T_4^{r_a} \cdot T_2^{r_f} \\
&= \hat{t}(A \cdot P_3^{a},Q)^{r_e} \cdot T_3^{r_{ae}} \cdot T_4^{r_a} \cdot T_2^{r_f} \\
&= \hat{t}(R,Q)^{r_e} \cdot T_2^{r_f} \cdot T_4^{r_a} \cdot T_3^{r_{ae}} \\
&= \hat{t}(R,Q)^{s_e + e \cdot c} \cdot T_2^{s_f - f \cdot c} \cdot T_4^{s_a - a \cdot c} \cdot T_3^{s_{ae} - a \cdot e \cdot c} \\
&= \hat{t}(R,Q)^{s_e} \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot (\hat{t}(R,Q)^{-e} \cdot T_2^{f} \cdot T_4^{a} \cdot T_3^{a \cdot e})^{-c} \\
&= \hat{t}(R,Q)^{s_e} \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot (\hat{t}(R,X) \cdot \hat{t}(R,X)^{-1} \cdot \\
&\quad \hat{t}(R,Q)^{-e} \cdot T_2^{f} \cdot T_4^{a} \cdot T_3^{a \cdot e})^{-c} \\
&= \hat{t}(R,Q)^{s_e} \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot (\hat{t}(R,X) \cdot \hat{t}(R,X \cdot Q^{e})^{-1} \cdot T_2^{f} \cdot T_4^{a} \cdot T_3^{a \cdot e})^{-c} \\
&= \hat{t}(R,Q)^{s_e} \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot (\hat{t}(R,X) \cdot \hat{t}(P_1^{1/(x+e)} \cdot \\
&\quad P_2^{f/(x+e)} \cdot P_3^{a}, X \cdot Q^{e})^{-1} \cdot T_2^{f} \cdot T_4^{a} \cdot T_3^{a \cdot e})^{-c} \\
&= \hat{t}(R,Q)^{s_e} \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot (\hat{t}(R,X) \cdot T_1^{-1} \cdot (T_2^{f} \cdot T_3^{a \cdot e} \cdot T_4^{a})^{-1}) \cdot \\
&\quad T_2^{f} \cdot T_4^{a} \cdot T_3^{a \cdot e})^{-c} \\
&= \hat{t}(R,Q)^{s_e} \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot (\hat{t}(R,X)/T_1)^{-c} \\
&= \hat{t}(R,Q^{s_e} \cdot X^{-c}) \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot T_1^{c}
\end{aligned}
$$

- Prior to running the protocol the Host decides if it wants σ to be linkable to other signatures produced for the same verifier. If it does not want the signature to be linkable to any existing or future signatures then it chooses $\mathsf{bsn} = \perp$. If it decides that it wants the signature to be linked to some previously generated signatures with this verifier then it sets bsn to be the same as

that used for the signature it wants to link to. Otherwise, if the Host decides it may want future signatures to be able to be link to this one then it chooses a verifier bsn that it has not used before. In some real DAA applications, the Host and Verifier should reach an agreement on what type of bsn and which bsn should be used in the Sign protocol.
- The use of J and K allows the verifier to identify if the signature was produced by a rogue TPM by computing J^{f_i} for all f_i values on the $\mathsf{RogueList}$ and comparing these to K. This check is performed during the verification algorithm.
- The Host is trusted to keep anonymity because it is assumed that the Host has the motivation to protect privacy and also because the host can always disclose the platform identity anyway. However, the Host is not trusted to be honest for not trying to forge a DAA signature without the aid of TPM.

3.4 The Verification Algorithm

This algorithm is run by a verifier $\mathfrak{v}$. Intuitively the verifier checks that a signature provided proves knowledge of a discrete logarithm f, checks that it proves knowledge of a valid credential issued by a given Issuer on the same value of f and that this value of f is not on the list of rogue values. We now describe the details of our Verify algorithm. On input of a signature $\sigma = (R, J, K, c, s_f, s_a, s_e, s_{ae})$, two nonces (n_V, n_T), a message msg, a basename bsn, an issuer public key $\mathsf{ipk} = X$ and the public system parameters par, this algorithm performs the following steps:

1. *Check Against* $\mathsf{RogueList}$. If $K = J^{f_i}$ for any f_i in the set of rogue secret keys then return *reject*.
2. *Check J computation.* If $\mathsf{bsn} \neq \perp$ and $J \neq H_3(\mathsf{bsn})$ then return *reject*.
3. *Verify Correctness of Proofs.* This is done by performing the following sets of computations:
   - $S' \leftarrow \hat{t}(R, Q^{s_e} \cdot X^{-c}) \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot T_1^{c}$.
   - $L' \leftarrow J^{s_f} \cdot K^{-c}$.
   - $\mathsf{str} \leftarrow X \| P_1 \| P_2 \| P_3 \| Q \| n_V$.
   - $h' \leftarrow H_4(\mathsf{str} \| R)$.

   Finally if $c \neq H_5\left(h' \| J \| K \| L' \| S' \| \mathsf{msg} \| n_T\right)$ return *reject* and otherwise return *accept*.

We note the verify algorithm ensures the bsn submitted with the signature is the one used by the TPM to compute K by checking J and K are correctly related to each other, and that $J = H_3(\mathsf{bsn})$ for $\mathsf{bsn} \neq \perp$.

3.5 The Linking Algorithm

This algorithm is run by a given verifier $\mathfrak{v}_j \in \mathfrak{V}$ which has a set of basenames $\{\mathsf{bsn}\}_j$ in order to determine if a pair of signatures were produced by the same

TPM. This algorithm is the same as in the DAA schemes of [13, 23]. Signatures can only be linked if they were produced by the same TPM and the user wanted them to be able to be linked together. Formally, on input a tuple $((\sigma_0, \mathsf{msg}_0), (\sigma_1, \mathsf{msg}_1), \mathsf{bsn}, \mathsf{ipk})$ the algorithm performs the following steps:

1. *Verify Both Signatures.* For each signature σ_b, for $b \in \{0, 1\}$ the verifier runs the algorithm $\mathsf{Verify}(\sigma_b,\ \mathsf{msg}_b,\ \mathsf{bsn},\ \mathsf{ipk})$ and if either of these returns *reject* then the value $\perp$ is returned.
2. *Compare J and K values.* If $J_0 = J_1$ and $K_0 = K_1$ then return *linked*, else return *unlinked*.

It may be the case that one or both signatures input to the Link algorithm have previously been received and verified by the verifier. Regardless of this we insist that the verifier re-verify these as part of the Link algorithm since the list of rogue TPM values may have been updated since the initial verification.

Also we note the condition that $K_0 = K_1$ ensures only signatures produced with the same basename *and* internal f value can be linked together. Since both signatures correctly verify with bsn this means that in each case the K and J values relate correctly to each other.

Note, our linking algorithm works due to the way that J and K are computed in the signing algorithm. Also note that anyone who knows bsn can link the two signatures, but they cannot link the signatures to the signers.

3.6 Revocation Consideration

In the literature, there are four types of revocation solutions are known for DAA. The first two solutions were proposed in the original DAA paper [11], whilst the third was proposed by Brickell and Li in [15] and the last one was proposed by Chen, Morrissey and Smart in [23]. These four solutions are summarized as follows:

1. Revocation is consequent upon a Signer's DAA secret becoming known. Anybody believing they have come into possession of a Signer's DAA secret (which, of course, should not happen) can put the secret into the rogue list RogueList and then check if this is truly the case by carrying out a check to verify whether a DAA signature from the Signer was signed using the secret in RogueList or not - if yes, the signature is rejected as the Signer's DAA secret has clearly been compromised. This solution works after a Signer's DAA secret is revealed.
2. A Verifier builds his own black list of unwelcome Signers. In order to find whether a DAA signature was signed by a black-listed Signer, the Verifier requires the Signer to use a specific basename in his DAA signature.
3. In each DAA signature, a Signer is required to prove, in a zero-knowledge proof manner, that his private signing key is not listed in a black list maintained by a revocation manager. The computation in the zero-knowledge proof requires possession of the private signing key, that means it needs to be done by the TPM rather than the host.

4. An Issuer updates his private and public keys at intervals, preferably regular; at each update of his keys the Issuer also correspondingly updates each DAA credential it holds unless, from the Issuer's knowledge, a Signer is no longer a legitimate DAA signer in which case the Issuer refuses to update the credential concerned. The Issuer publishes his updated public key and makes each updated DAA credential available to the corresponding Signer. In this solution, there is no extra cost to the TPM in the DAA signing algorithm. The key updating process is transparent to the TPM.

Our new DAA scheme is suitable for all of these revocation solutions. Choice of them is dependent upon applications. This feature has no difference from the existing DAA schemes. So we do not discuss them further in this paper.

4 Security Proof of the DAA Scheme

In this section, we will state the security results for the new DAA scheme under the definitions of security notions in Section 2.1. In general, we will argue that our new DAA scheme is secure, i.e., correct, user-controlled-anonymous and user-controlled-traceable, as addressed in the following theorems.

Our security results are based on the q-SDH assumption and the $\mathbb{G}_1$-DDH assumption as defined in Section 2.2. The security analysis of the notions of user-controlled-anonymity and user-controlled-traceability is in the random oracle model [6], i.e., we will assume that the hash functions H_2 H_3 and H_5 in the new DAA scheme are random oracles. Note that the hash function H_1 used to compute the value f and H_4 used in the Sign protocol do not have to be random oracles, since they are internal functions.

Theorem 1. *The DAA scheme specified in Section 3 is correct.*

Proof. This theorem follows directly from the specification of the scheme. $\square$

Theorem 2. *Under the $\mathbb{G}_1$-DDH assumption in Definition 5, the above DAA scheme is user-controlled-anonymous. More specifically, if there is an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break user-controlled-anonymity of the scheme, then there is a simulator $\mathcal{S}$ running in polynomial time that solves the $\mathbb{G}_1$-DDH problem with a non-negligible probability.*

Proof. We will show how an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break user-controlled-anonymity of the DAA scheme may be used to construct a simulator $\mathcal{S}$ that solves the $\mathbb{G}_1$-DDH problem. Let $(P, P^{\mathsf{a}}, P^{\mathsf{b}}, P^{\mathsf{c}}) \in \mathbb{G}_1^4$ and $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathbb{Z}_p^*$ be the instance of the $\mathbb{G}_1$-DDH problem that we wish to answer whether P^{c} is equal to P^{ab} or not. We now describe the construction of the simulator $\mathcal{S}$, which performs the following game with $\mathcal{A}$, as defined in Section 2.1.

In the initial of the game, $\mathcal{S}$ runs `Setup` (or takes $\mathcal{A}$'s input) to create an issuer, which is named by an identifier K_I and which is of two issuer public keys, say $\mathfrak{i}_0$ and $\mathfrak{i}_1$. Each public key is presented as $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, \hat{t}, P_1, P_2, P_3, Q, T_1, T_2, T_3, T_4, H_1, H_2, H_3, H_4, H_5, \mathsf{ipk} = Q^x)$ and its corresponding secret key is presented as $\mathsf{isk} = x$. All the values of the public and secret keys are known to $\mathcal{A}$. For the purpose of simplicity, we do not ask these two keys to be completely different to each other. We only assume that these two public keys have different P_2 values and their H_3 functions are each relevant to their P_2 values. Note that actually it does not matter if some other values between these two keys are different from each other, although it is not required for the purpose of our proof. Throughout the proof, except for some individual specification, we do not use different notation to distinguish these two keys.

More specifically, in the first key for $\mathfrak{i}_0$, $P_2 = P$, and in the second one for $\mathfrak{i}_1$, $P_2 = P^{\mathsf{b}}$. In both of the public keys, $H_3(w) = P_2^{r_w} \in \mathbb{G}_1$, where r_w is chosen uniformly at random in $\mathbb{Z}_p^*$. Note that since these two H_3 functions make use of different P_2 as the base, so for the same input w value, their outputs of H_3 are different to each other. Throughout the proof specification, $/\mathfrak{i}_b$, where $b = \{0, 1\}$, indicates which issuer's public key is associated with.

$\mathcal{S}$ creates algorithms to respond to queries made by $\mathcal{A}$ during its attack, including three random oracles denoted by H_2, H_3 and H_5 in the DAA scheme.

To maintain consistency between queries made by $\mathcal{A}$, $\mathcal{S}$ keeps the following lists: L_i for $i = 2, 3, 5$ stores data for query and response pairs to random oracle H_i. L_{jc} stores data for query and response records for Join queries and Corrupted queries. Each item of L_{jc} is $\{ID/\mathfrak{i}_b, f, F, \mathsf{cre}, c\}$, where $c = 1$ means that the corresponding signer is corrupted and $c = 0$ otherwise. L_s stores data for query and response records for Sign queries. Each item of L_s is $\{ID/\mathfrak{i}_b, m, \mathsf{bsn}, \sigma, s\}$, where $s = 1$ means that $\mathsf{bsn} = \perp$ and $s = 0$ means that $\mathsf{bsn} \neq \perp$ under the Sign query. At the beginning of the simulation, $\mathcal{S}$ sets all the above lists empty. An empty item is denoted by the symbol *. During the game, $\mathcal{A}$ will asks the H_i queries up to q_i times, asks the Join query up to q_j times, asks the Corrupt query up to q_c times, and asks the Sign query up to q_s times. All of these time values are polynomial.

Simulator: $H_2(m)$**.** If $(m, h_2) \in L_2$, return h_2. Else choose h_2 uniformly at random from $\mathbb{Z}_p^*$; add (m, h_2) to L_2 and return h_2.

Simulator: $H_3(m)/\mathfrak{i}_b$**.** If m has already been an entry of the $H_3/\mathfrak{i}_b$ query, i.e. the item $(m, w, h_3/\mathfrak{i}_b)$ for an arbitrary w and $h_3/\mathfrak{i}_b$ exists in L_3, return $h_3/\mathfrak{i}_b$. Else choose v from $\mathbb{Z}_p^*$ uniformly at random; compute $h_3/\mathfrak{i}_b \leftarrow P_2^v$; add $(m, v, h_3/\mathfrak{i}_b)$ to L_3 and return $h_3/\mathfrak{i}_b$.

Simulator: $H_5(m)$**.** If $(m, h_5) \in L_5$, return h_5. Else choose h_5 uniformly at random from $\mathbb{Z}_p^*$; add (m, h_5) to L_5 and return h_5.

Simulator: Join(ID). At the beginning of the simulation choose α, β uniformly at random from $\{1, ..., q_j\}$. We show how to respond to the i-th query made by $\mathcal{A}$ below. Note that we assume $\mathcal{A}$ does not make repeat queries, but we also assume that for each query, the Join protocol could be run twice, one with i_0 and the other with i_1. Although it seems redundant for the query of every ID to be run twice, it is necessary for $i = \alpha$ or β. We use ID_X/i_b, $b \in \{0,1\}$, to indicate the singer identity ID_X associated with i_b.

- If $i = \alpha$ and in the run associated with i_0, set $F_\alpha \leftarrow P^{\mathsf{a}}$ (i.e. P_2^{a}); run $\mathsf{Join_t}$ with $\mathcal{A}$ to get cre_α, and add $\{ID_\alpha/\mathsf{i}_0, *, F_\alpha, \mathsf{cre}_\alpha, 0\}$ to L_{jc}. Note that since $\mathcal{S}$ does not know the value $f_\alpha = \mathsf{a}$ (which is indicated as $*$ in L_{jc}), it is not able to compute (v, w) by following the Schnorr signature scheme. However $\mathcal{S}$ can forge the signature by controlling the random oracle of H_2 as follows: randomly choose w and v and compute $U = P_2^w F_\alpha^{-v}$. The only thing $\mathcal{S}$ has to take care of is checking the consistency of the L_2 entries. $\mathcal{S}$ verifies the validation of cre_α before accepting it.
- If $i = \beta$ and in the run associated with i_1, set $F_\beta \leftarrow P^{\mathsf{c}}$ (i.e. $P_2^{\mathsf{c}/\mathsf{b}}$); do the same thing as in the previous item to get cre_β.
- Else, including $i = \alpha$ with i_1 and $i = \beta$ with i_0, choose f uniformly at random from $\mathbb{Z}_p^*$; compute $F = P_2^f$, if $F = P_2^{\mathsf{a}}$ or P_2^{b}, abort outputting "**abortion 0**"; else run $\mathsf{Join_t}$ with $\mathcal{A}$ to get cre; verify cre before accept it and then add $(ID/\mathsf{i}_b, f, F, \mathsf{cre}, 0)$ in L_{jc}.

Simulator: Corrupt(ID). We assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Corrupt query using the identity. Otherwise, $\mathcal{S}$ answers the Join query first. Find the entry $(ID/\mathsf{i}_b, f, F, \mathsf{cre}, 0)$ in L_{jc}, return f and update the item to $(ID/\mathsf{i}_b, f, F, \mathsf{cre}, 1)$.

Simulator: Sign(ID, m, bsn). Let m' be the input message $\mathcal{A}$ wants to sign, $n_V \in \{0,1\}^t$ be a nonce chosen by $\mathcal{A}$ and $n_T \in \{0,1\}^t$ be a nonce chosen by $\mathcal{S}$ at random, so $m = (m', n_V, n_T)$. We assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Sign query using the identity. Otherwise, $\mathcal{S}$ answers the Join query first. We have the following multiple cases to consider.

Case 1: $ID/\mathsf{i}_b \neq ID_\alpha/\mathsf{i}_0$ and $ID/\mathsf{i}_b \neq ID_\beta/\mathsf{i}_1$. Find the entry $(ID/\mathsf{i}_b,\ f,\ F,$ $\mathsf{cre},\ 0/1)$ in L_{jc}, compute $\sigma \leftarrow \mathsf{Sign}$, add $(ID/\mathsf{i}_b, m, \mathsf{bsn}, \sigma, 0/1)$ to L_s and respond with σ.

Case 2: $ID/\mathsf{i}_b = ID_\alpha/\mathsf{i}_0$. $\mathcal{S}$ is not able to create such a signature since $\mathcal{S}$ does not know the corresponding secret key. But $\mathcal{S}$ is able to forge the signature by controlling the random oracles of H_3 and H_5. $\mathcal{S}$ finds the entry $(ID_\alpha/\mathsf{i}_0, *, F_\alpha, \mathsf{cre}_\alpha = (A_\alpha, e_\alpha), 0)$ in L_{jc}, and forges σ by performing the following steps:

1. When $\mathsf{bsn} = \perp$, choose a random r; search whether r is an entry of L_3; if yes, go back to the beginning of this item. When $\mathsf{bsn} \neq \perp$, take the given bsn, search whether bsn is an entry of L_3; if yes, retrieve the corresponding v and $h_3 = P_2^v$. With a new input of L_3, query H_3 to get v and h_3.
2. Set $J \leftarrow h_3 = P^v$ and $K \leftarrow (P^{\mathsf{a}})^v$.
3. Choose random $a \leftarrow \mathbb{Z}_p^*$ and compute $R \leftarrow A_\alpha \cdot P_3^a$.

4. Compute $\mathsf{str} \leftarrow X \| P_1 \| P_2 \| P_3 \| Q \| n_v$.
5. Choose $s_f, s_a, s_e, s_{ae} \in \mathbb{Z}_p^*$ at random.
6. Choose c at random; search whether c is an entry of L_5; if yes, go back to the beginning of this item.
7. Compute $S \leftarrow \hat{t}(R, Q^{s_e} \cdot X^{-c}) \cdot T_2^{s_f} \cdot T_4^{s_a} \cdot T_3^{s_{ae}} \cdot T_1^c$.
8. Compute $L \leftarrow J^{s_f} \cdot K^{-c}$.
9. Set $w = H_4(\mathsf{str} \| R) \| J \| K \| L \| S \| m \| n_T$. Search whether w is an entry of L_5; if yes, go back to the beginning of the item of choosing s_f, s_a, s_e, s_{ae}; otherwise, add (w, c) in L_5.
10. Output $\sigma = (R, J, K, c, s_f, s_a, s_e, s_{ae})$.
11. Add $(ID_\alpha / \mathsf{i}_0, m, \mathsf{bsn}, \sigma, 1/0)$ to L_s.

Case 3: $ID/\mathsf{i}_b = ID_\beta/\mathsf{i}_1$. Again, $\mathcal{S}$ cannot create this signature properly without the knowledge of f_β. $\mathcal{S}$ forges the signature in the same way as in Case 2 above, except setting $J = h_3 = P_2^v = (P^{\mathsf{b}})^v$ and $K = (P^{\mathsf{c}})^v$.

At the end of Phase 1, $\mathcal{A}$ outputs a message m, a basename bsn, two identities $\{ID_0, ID_1\}$. If $\{ID_0, ID_1\} \neq \{ID_\alpha, ID_\beta\}$, $\mathcal{S}$ aborts outputting "**abortion 1**". We assume that Join has already been queried at ID_α and ID_β by $\mathcal{A}$ associated with both i_0 and i_1. If this is not the case we can define Join at these points as we wish i.e. as for ID_α/i_0, $F_\alpha = P^{\mathsf{a}}$ and for ID_β/i_1, $F_\beta = P^{\mathsf{c}}$. Neither ID_0 nor ID_1 should have been asked for the Corrupt query and the Sign query with the same $\mathsf{bsn} \neq \perp$ by following the definition of the game defined in Section 2.1.

$\mathcal{S}$ chooses a bit b at random, and generates the challenge in the same way as Case 2 or 3 of the Sign query simulation, by querying Sign(ID_α, m, bsn) with i_0 if $b = 0$ or Sign(ID_β, m, bsn) with i_1 otherwise. $\mathcal{S}$ returns the result σ^* to $\mathcal{A}$.

In Phase 2, $\mathcal{S}$ and $\mathcal{A}$ carry on the query and response process as in Phase 1. Again, $\mathcal{A}$ is not allowed to make any Corrupt query to either ID_0 or ID_1 and to make any Sign query to either ID_0 or ID_1 with the same $\mathsf{bsn} \neq \perp$. At the end of Phase 2, $\mathcal{A}$ outputs b'. If $b' = b$, $\mathcal{S}$ outputs 0, which means $P^{\mathsf{c}} \neq P^{\mathsf{ab}}$; otherwise $\mathcal{S}$ outputs 1, which means $P^{\mathsf{c}} = P^{\mathsf{ab}}$.

Let ϵ be the probability that A succeeds in breaking the anonymity game. Suppose $\mathcal{S}$ does not abort during the above simulation. If $\mathsf{c} \neq \mathsf{ab}$, $\mathcal{S}$ emulates the anonymity game perfectly, i.e., $Pr[b = b'] = 1/2 + \epsilon$. If $\mathsf{c} = \mathsf{ab}$, then the private keys for ID_0/i_0 and ID_1/i_1 are identical and thus the signature σ^* is independent of b. It follows that $Pr[b = b'] = 1/2$. Therefore, assuming $\mathcal{S}$ does not abort, it has advantage at least $\epsilon/2$ in solving the $\mathbb{G}_1$-DDH problem.

We can argue that creating two issuer public keys in the game does not make the simulation distinguishable from the real DAA scheme. In the formal definition of DAA specified in Section 2.1, a system can involves multiple issuers, signers and verifiers; each signer can obtain multiple DAA credentials associated with the same DAA secret. For the flexibility, the signer's DAA secret f is relevant to the issuer's identifier K_I, which could be the issuer's root public key as specified in [38] or the issuer's partial public parameters which is used by the TPM. In our proof, we only require that the two issuer public keys are associated with the same K_I value, and a single TPM DAA secret f could naturally be computed and then associated with the two different issuer pubic keys.

Therefore the adversary $\mathcal{A}$ should not be able to notice any difference between the real DAA scheme and the simulation based on the double issuer public keys.

Let us now consider how our simulation could abort i.e. describe events that could cause $\mathcal{A}$'s view to differ when run by $\mathcal{S}$ from its view in a real attack.

It is clear that the simulations for H_2, H_3 and H_5 are indistinguishable from real random oracles.

If the event **abortion 0** happens, $\mathcal{S}$ gets the value a or b, $\mathcal{S}$ can compute P^{ab} and thus to solve the DDH problem (because the DDH problem is weaker than the CDH problem). Since $\mathcal{S}$ chooses its value uniformly at random from $\mathbb{Z}_p^*$, the chance of this event happening is negligible.

The event **abortion 1** happens only if $\{ID_0, ID_1\} \neq \{ID_\alpha, ID_\beta\}$. Since ID_α and ID_β are chosen at random, the probability of this case is at least $1/(q_j(q_j-1))$.

Based on the above discussion, the probability that $\mathcal{S}$ does not abort the game at some stage and produces the correct output is non-negligible, since it follows the fact that $\mathcal{A}$ wins the game with a non-negligible probability. $\square$

Theorem 3. *Under the q-SDH assumption, the above DAA scheme is user-controlled-traceable. More specifically, if there is an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break user-controlled-traceability of the scheme, then there is a simulator $\mathcal{S}$ running in polynomial time that solves the q-SDH problem with a non-negligible probability.*

Proof. We will show how an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break user-controlled-traceability of the DAA scheme may be used to construct a simulator $\mathcal{S}$ that solves the q-SDH problem. Let $(P', Q', Q'^x, Q'^{x^2}, ..., Q'^{x^q}) \in \mathbb{G}_1 \times \mathbb{G}_2^{q+1}$ be the instance of the q-SDH problem. We let $\mathcal{S}$ performs the user-controlled-traceability game as specified in Section 2.1 with $\mathcal{A}$ twice. Each performance is corresponding to one of the two initial cases. We now describe the construction of the simulator $\mathcal{S}$ in these two performances one by one.

In the first performance, we apply the technique used in the proof of Boneh and Boyen's Lemma 1 [8], obtaining generators $P_1 \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, $X = Q^x$, and $q-1$ SDH pairs (B_i, e_i) such that $\hat{t}(B_i, X \cdot Q^{e_i}) = \hat{t}(P_1, Q)$ for each i. We wish to provide one more SDH pair (B', e') besides these $q-1$ pairs that can be transformed into a solution to the original q-SDH instance, again based on Boneh and Boyen's proof of Lemma 1.

$\mathcal{S}$ performs the following game with $\mathcal{A}$. In the initial of the game, $\mathcal{S}$ sets the system public parameters par as $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, \hat{t},\ P_1, P_2, P_3,\ Q, T_1, T_2, T_3, T_4, H_1, H_2,\ H_3, H_4, H_5, \mathsf{ipk})$ as follows: randomly select three random numbers $a, b, e \in \mathbb{Z}_p^*$, set $P_1 = \psi(Q)$ and $P_2 = \psi([(Q^x \cdot Q^e)^b \cdot Q^{-1}]^{1/a})$. Note that $P_2 = P_1^{((x+e)b-1)/a}$ holds. $\mathcal{S}$ also sets $\mathfrak{I}$'s public key $\mathsf{ipk} = X = Q^x$ and the corresponding secret key, namely isk, as x. $\mathcal{S}$ gives par to $\mathcal{A}$, but keeps the values a, b, e for itself. Note that $\mathcal{S}$ does not know isk. It also creates algorithms to respond to queries made by $\mathcal{A}$ during its attack.

$\mathcal{S}$ sets three random oracles H_2, H_3 and H_5 in the ordinary way. To maintain consistency between queries made by $\mathcal{A}$, $\mathcal{S}$ keeps the following lists: L_i

for $i = 2, 3, 5$ stores data for query and response pairs to random oracle H_i. L_{jc} stores data for query and response records for Join queries and Corrupted queries. Each item of L_{jc} is $\{ID, f, F, \mathsf{cre}, c\}$, where $c = 1$ means that the corresponding signer is corrupted (via either Case 2 of the Join query or the Corrupt query) and $c = 0$ otherwise. Note that the set of f values with $c = 1$ will be used as the $\mathsf{RogueList}$ list. L_s stores data for query/response records for Sign queries. Each item of L_s is $\{ID, m, \mathsf{bsn}, \sigma, s\}$, where $s = 1$ means that $\mathsf{bsn} = \perp$ under the Sign query and $s = 0$ means that $\mathsf{bsn} \neq \perp$ under the Sign query. At the beginning of the simulation, $\mathcal{S}$ sets all the above lists empty. An empty item is denoted by the symbol *. During the game, $\mathcal{A}$ will asks the H_i queries up to q_i times, asks the Join query up to q times, asks the Corrupt query up to q_c times, and asks the Sign query up to q_s times. All of the time values are polynomial.

Simulator: $H_2(m)$**.** The same as in the proof of Theorem 2.

Simulator: $H_3(m)$**.** If m has already been an entry of the H_3 query, return h_3. Else choose h_3 from $\mathbb{Z}_p^*$ uniformly at random and return h_3.

Simulator: $H_5(m)$**.** The same as in the proof of Theorem 2.

Simulator: Join(ID**).** $\mathcal{A}$ allows to make upon to q Join queries. We assume $\mathcal{A}$ does not make repeat queries. As defined in the game of user-controlled-traceability, there are two Join cases associated with the Initial Case 1. In the Join Case 1, given a new ID from $\mathcal{A}$, $\mathcal{S}$ returns the credential cre for the value f, and adds $\{ID, f, F, \mathsf{cre}, 0\}$ to L_{jc}. In the Join Case 2, $\mathcal{S}$ receives a new pair of ID and f from $\mathcal{A}$, returns cre and adds $\{ID, f, F, \mathsf{cre}, 1\}$ to L_{jc}. $\mathcal{S}$ randomly selects one query from the Join Case 1 and treats it specially by letting $f = a$ and $\mathsf{cre} = (A, e)$ where $A = P_1^b$. In the other times of the Join Case 1, $\mathcal{S}$ selects the value f_i at random. In each of the remaining $q - 1$ queries including both the Join Case 1 and Case 2, $\mathcal{S}$ makes use of one SDH pair (B_i, e_i) to compute $\mathsf{cre}_i = (A_i, e_i)$ for f_i, where

$$A_i = (P_1 \cdot P_2^{f_i})^{1/(x+e_i)} = B_i^{1-f_i/a+f_ib(e-e_i)/a} \cdot P_1^{f_ib/a}.$$

Simulator: Corrupt(ID**).** We assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Corrupt query using the identity. Otherwise, $\mathcal{S}$ answers the Join query first. Find the entry $(ID, f, F, \mathsf{cre}, 0)$ in L_{jc}, return f and update the item to $(ID, f, F, \mathsf{cre}, 1)$.

Simulator: Sign(ID, m, bsn**).** Let m' be the input message $\mathcal{A}$ wants to sign, $n_V \in \{0,1\}^t$ be a nonce chosen by $\mathcal{A}$ and $n_T \in \{0,1\}^t$ be a nonce chosen by $\mathcal{S}$ at random, so $m = (m', n_V, n_T)$. We assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Sign query using the identity. Otherwise, $\mathcal{S}$ answers the Join query (Case 1) first. Find the entry $(ID, f, F, \mathsf{cre}, 0/1)$ in L_{jc}, compute $\sigma \leftarrow \mathsf{Sign}$. In the end, $\mathcal{S}$ adds $(ID, m, \mathsf{bsn}, \sigma, 1/0)$ to L_s and responds with σ.

Simulator: Semi-sign($ID, m, J, \hat{S}, h$). We assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Semi-sign query using the identity. Otherwise, $\mathcal{S}$ answers the Join query (Case 1) first. Find the entry $(ID, f, F, \mathsf{cre}, 0/1)$ in L_{jc}, compute (K, c, s_f) by following TPM's action in Sign, add $(ID, m, \mathsf{bsn}, \sigma = (*, J, K, c, s_f, *, *, *), 1/0)$ to L_s and respond with (K, c, s_f).

At the end of the phase of probing above, $\mathcal{A}$ outputs an identity ID^*, a message m^*, a basename bsn^* and a signature σ^*. We consider the following two cases:

- Case 1. If $\mathsf{Verify}(\sigma^*) = 1$ and $(ID^*, m^*, \mathsf{bsn}^*, \sigma^*, 1/0)$ (or $(ID^*, m^*, \mathsf{bsn}^*, \sigma^* = (*, J, K, c, s_f, *, *, *), 1/0)$ is not in L_s, $\mathcal{S}$ rewinds $\mathcal{A}$ to extract the knowledge of cre^* and f^*, satisfying $\mathsf{cre}^* = (A^*, e^*)$ is a BBS signature on the message f^*. To rewind $\mathcal{A}$, $\mathcal{S}$ controls the challenge c by choosing two different c values c' and c'' to the same R and S. $\mathcal{A}$ will responds them with $s'_f, s'_a, s'_e, s'_{ae}$ and $s''_f, s''_a, s''_e, s''_{ae}$ respectively. $\mathcal{S}$ then computes $\Delta_c = c' - c''$ and extracts $f^* = (s'_f - s''_f)/\Delta_c$, $e^* = (s'_e - s''_e)/\Delta_c$, $a^* = (s'_a - s''_a)/\Delta_c$ and $A^* = R/(P_3^{a^*})$. Since the value $f^* \notin \mathsf{RogueList}$ (implied in $\mathsf{Verify}(\sigma) = 1$), there are the following two possible results:
  1. If $e^* \ni \{e_i, e\}$ for any i, $\mathcal{S}$ computes
  $$B^* = (A^* \cdot P_1^{-bf^*/a})^{a/(a-f^*+bf^*(e-e^*))}.$$
  2. If $e^* \in \{e_i, e\}$ and $A^* \ni \{A_i, A\}$ for any i, with the probability of $1/q$, $e^* = e$, $\mathcal{S}$ computes
  $$B^* = (A^* \cdot P_1^{-bf^*/a})^{a/(a-f^*)}.$$

  Observe that $\mathcal{S}$ is able to create more than q "valid" copies of the DAA secret f and its credential cre. For example, from any pair of (B_i, e_i) where $i = \{1, 2, ..., q-1\}$, $\mathcal{S}$ can randomly choose two f_{i0}, f_{i1} values and computes two corresponding A_{i0}, A_{i1} values, and then obtains two valid triples (f_{i0}, A_{i0}, e_i) and (f_{i1}, A_{i1}, e_i). The triple $(f = a,\ A = P_1^b,\ e)$ is a special case for $\mathcal{S}$ but not for $\mathcal{A}$. From $\mathcal{A}$'s point of view, this case is indistinguishable from the other $q-1$ cases in the total q Join queries, since $\mathcal{A}$ does not know the values of a and b, and does not know the index of (f, A, e). We can argue that if $\mathcal{A}$ is able to create (f_{j0}, f_{j1}) and (A_{j0}, A_{j1}) with the same e_j for any $j = \{1, 2, ..., q\}$ (rather than the total $q-1$ values of i for $\mathcal{S}$), then the probability of choosing the j value should be equal to $1/q$.

  In either of these two results, $\mathcal{S}$ takes (B^*, e^*) as the extra SDH pair to solve the given q-SDH problem.
- Case 2. Suppose $\mathsf{bsn}^* \neq \perp$. If there is no any entry $(ID^*, m', \mathsf{bsn}^*, \sigma', 0)$ for the arbitrary pair of m' and σ' is found in L_s, $\mathcal{A}$ has not managed to break user-controlled-traceability. Otherwise, $\mathcal{S}$ runs $\mathsf{Link}(\sigma^*, \sigma')$. If the

output of Link is 1 or $\perp$, again, $\mathcal{A}$ has not managed to break user-controlled-traceability. Otherwise, there exist the following pair of data sets $\sigma^* = (R, J, K, c, s_f, s_a, s_e, s_{ae})$ and $\sigma' = (R', J', K', c', s'_f, s'_a, s'_e, s'_{ae})$. $J = J'$ holds since two signatures have the same bsn and $\mathcal{S}$ has maintained the consistency of the random oracle H_3 outputs. The only thing to make $K \neq K'$ happen is that $\mathcal{A}$ has managed to create a different tsk for ID. Then $\mathcal{S}$ can use the same trick as in Case 1 to extract a right solution of the q-SDH problem from $\mathcal{A}$.

In either of the above two cases, $\mathcal{S}$ can solve the q-SDH problem with a non-negligible probability if $\mathcal{A}$ wins the game with a non-negligible probability.

In the second performance, $\mathcal{S}$ performs the following game with $\mathcal{A}$. In the initial of the game, $\mathcal{S}$ sets the system public parameters par as $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, \hat{t}, P_1, P_2, P_3, Q, T_1, T_2, T_3, T_4, H_1, H_2, H_3, H_4, H_5, \mathsf{ipk})$ as follows: randomly select a number $r \in \mathbb{Z}_p^*$, set $Q = Q'$, $P_2 = \psi(Q)$ and $P_1 = P_2^r$. $\mathcal{S}$ sends $\mathcal{A}$ the values of par and receives $\mathfrak{I}$'s public key $\mathsf{ipk} = X = Q^x$ from $\mathcal{A}$. Note that the value x, namely isk, is not known to $\mathcal{S}$, although $\mathcal{S}$ does verify that the value X is in the right group $\mathbb{G}_2$.

$\mathcal{S}$ also creates algorithms to respond to queries made by $\mathcal{A}$ during its attack. $\mathcal{S}$ sets three random oracles H_2, H_3 and H_5, maintains consistency between queries made by $\mathcal{A}$, and keeps the lists of L_i (for $i = 2, 3, 5$), L_{jc} and L_s in the same way as in Performance 1. During the game, $\mathcal{A}$ will asks the H_i queries up to q_i times, asks the Join query up to q_j times, asks the Corrupt query up to q_c times, and asks the Sign query up to q_s times. All of the time values are polynomial.

Simulator: $H_2(m)$**.** The same as in the proof of Theorem 2.

Simulator: $H_3(m)$**.** If m has already been an entry of the H_3 query, i.e. the item (m, w, h_3) for an arbitrary w and h_3 exists in L_3, return h_3. Else choose v from $\mathbb{Z}_p^*$ uniformly at random; computer $h_3 \leftarrow P_2^v$; add (m, v, h_3) to L_3 and return h_3.

Simulator: $H_5(m)$**.** The same as in the proof of Theorem 2.

Simulator: Join(ID**).** At the beginning of the performance, choose α uniformly at random from $\{1, ..., q_j\}$. We show how to respond to the i-th query made by $\mathcal{A}$ below. Note that we assume $\mathcal{A}$ does not make repeat queries.

– If $i = \alpha$, set $F_\alpha \leftarrow \psi(Q'^x)$; run $\mathsf{Join_t}$ with $\mathcal{A}$ to get cre_α, and add $\{ID_\alpha, *, F_\alpha, \mathsf{cre}_\alpha, 0\}$ to L_{jc}. Note that since $\mathcal{S}$ does not know the value $f_\alpha = x$ (which is indicated as $*$ in L_{jc}), it is not able to compute (v, w) by following the Schnorr signature scheme. However $\mathcal{S}$ can forge the signature by controlling the random oracle of H_2 as the same as it did in the proof of Theorem 2. $\mathcal{S}$ verifies the validation of cre_α before accepting it.

– Else choose f uniformly at random from $\mathbb{Z}_p^*$; compute $F \leftarrow P_2^f$, if $F = \psi(Q'^x)$, abort outputting "**abortion 0**"; else run $\mathsf{Join_t}$ with $\mathcal{A}$ to get cre; verify cre before accept it and then add $(ID, f, F, \mathsf{cre}, 0)$ into L_{jc}.

Simulator: Corrupt(ID). The same as in Performance 1.

Simulator: Sign(ID, m, bsn). We assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Sign query using the identity. Otherwise, $\mathcal{S}$ first answers the Join query (as described before for this performance). Find the entry $(ID, f, F,$ cre, $0/1)$ in L_{jc}. If $ID \neq ID_\alpha$, compute $\sigma \leftarrow \mathsf{Sign}$; otherwise, $\mathcal{S}$ does not know the value f, and it forges a σ using the same techniques as in Theorem 2. In the end, $\mathcal{S}$ adds $(ID, m, \mathsf{bsn}, \sigma, 1/0)$ to L_s and responds with σ.

Simulator: Semi-sign($ID, m, J, \hat{S}, h$). Again, we assume that $\mathcal{A}$ makes the queries Join(ID) before it makes the Semi-sign query using the identity. Otherwise, $\mathcal{S}$ answers the Join query first. Find the entry $(ID, f, F, \mathsf{cre}, 0/1)$ in L_{jc}. If $ID \neq ID_\alpha$, compute (K, c, s_f) by following TPM's action in Sign, otherwise, $\mathcal{S}$ does not know the value f, and it forges the triple (K, c, s_f) using the same techniques as it forges the signature σ. Add $(ID, m, \mathsf{bsn}, \sigma = (*, J, K, c, s_f, *, *, *),$ $1/0)$ to L_s and respond with (K, c, s_f).

At the end of the phase of probing above, $\mathcal{A}$ outputs an identity ID^*, a message m^*, a basename bsn^* and a signature σ^*. If $ID \neq ID_\alpha$, $\mathcal{S}$ aborts outputting "**abortion 1**". Otherwise, we consider the following situation:

If $\mathsf{Verify}(\sigma^*) = 1$ and $(ID^*, m^*, \mathsf{bsn}^*, \sigma^*, 1/0)$ (or $(ID^*, m^*, \mathsf{bsn}^*, \sigma^* = (*, J, K,$ $c, s_f, *, *, *), 1/0)$ is not in L_s, $\mathcal{S}$ rewinds $\mathcal{A}$ to extract the knowledge of f^*. To rewind $\mathcal{A}$, $\mathcal{S}$ uses the same technique as in Performance 1 by choosing two different c values c' and c'' to the same R and S. $\mathcal{A}$ will responds them with $s'_f, s'_a, s'_e, s'_{ae}$ and $s''_f, s''_a, s''_e, s''_{ae}$ respectively. $\mathcal{S}$ then computes $\Delta_c = c' - c''$ and extracts $f^* = (s'_f - s''_f)/\Delta_c$. $\mathcal{S}$ treats the value f as the value x in the given q-SDH problem. Since the q-SDH problem is weaker than the discrete logarithm problem, $\mathcal{S}$ in possession of the value x can easily find a new SDH pair $(e, P'^{1/(x+e)})$ by choosing a random e in $\mathbb{Z}_p^*$.

By following the same discussion in the proof of Theorem 2, we can show how our simulation could only abort with reasonably small probabilities, since both **abortion 0** and **abortion 1** are similar to these two abortions in the previous proof.

In either of the above two performances, $\mathcal{S}$ can solve the q-SDH problem with a non-negligible probability if $\mathcal{A}$ wins the game with a non-negligible probability. The theorem follows. □

5 Performance Comparison

In this section, we compare efficiency of the proposed DAA scheme with all the existing ECC-DAA schemes, based on our best knowledge. We do not include RSA-DAA schemes such as these in [11, 27], since the comparison between the

RSA-based schemes and pairing-based schemes has been presented in a number of papers. In general speaking, we see that the ECC-DAA schemes are a lot more efficient than the one based on factoring. We refer to [13, 21, 23, 24] for the detailed information.

In Table 1, we present some performance figures for the six ECC-DAA schemes. For the computational cost, we consider the Join protocol, Sign protocol and Verify algorithm, with respect to each player. We do not specify the computational cost of the Setup algorithm and its verification, since this is only run once and the resulting parameters are only verified once by each part. We do not specify the cost for the linking algorithm either, as it is closely related to that of the verification algorithm. For the communication cost and storage cost, we consider the credential size and signature size, but ignore the size of TPM secret key (i.e. the values of f and DAAseed) because this could be the same in all of the six schemes.

In this table, for the computational cost, we let $\mathbb{G}_i$ $(i = \{1, 2, T\})$ denote the cost of an exponentiation in the group $\mathbb{G}_i$, and $\mathbb{G}_i^m$ denote the cost of a multi-exponentiation of m values in the group $\mathbb{G}_i$. Note, that a multiexponentiation with m exponents can often be performed significantly faster than m separate exponentiations, which is why we separate this out. We also let P denote the cost of a pairing computation. In addition in the table we let n denote the number of keys in the verifier's rogue secret key list. For the credential and signature sizes, we let p denote the size of the prime order p, $\mathbb{G}_i$ $(i = \{1, 2, T\})$ denote the size of an element of the group $\mathbb{G}_i$, G denote the size of an element of the group G, which is used in [13, 16, 24] as a group that might be separated from $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, and h denote an output of a hash-function used in the Schnorr-type signature schemes [36]. The Brickell et al. scheme [13] uses symmetric pairings $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_T$, and the other five schemes in [16, 21, 23, 24] and this paper use asymmetric pairings $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$.

We recall the argument made in [23] regarding the pairing implementation. In the implementation of symmetric pairings $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_T$, the operations in $\mathbb{G}_1$ are about 1/4 the cost of operations in $\mathbb{G}_T$. This is a rough estimate derived as follows: At best $\mathbb{G}_T$ is a subgroup of $\mathbb{F}_{q^6}$ and operations in $\mathbb{F}_q$ will be $36 = 6^2$ times more efficient generally than operations in $\mathbb{G}_T$, $\mathbb{G}_1$ is an elliptic curve over $\mathbb{F}_q$ and so will have operations which take around 10 $\mathbb{F}_q$ operations, and $10/36 \approx 1/4$. By using asymmetric pairings $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$, one can use highly efficient curve choices such as Barreto-Naehrig curves [4]. We are thus able to obtain, for the same size of $\mathbb{G}_T$, operations in $\mathbb{G}_1$ which are around $144/10 \approx 14$ times more efficient than those in $\mathbb{G}_T$, as opposed to 4 times as above. This is because now $\mathbb{G}_T$ is a subgroup of $\mathbb{F}_{q^{12}}$. Using Barreto-Naehrig curves and sextic twists the operations in $\mathbb{G}_2$ will cost roughly four times those in $\mathbb{G}_1$, since $\mathbb{G}_2$ will be a subgroup of an elliptic curve defined over the field $\mathbb{F}_{q^2}$.

Observe that in [24], the rogue ragging operation is not defined in the Verify algorithm, but it can be easily added in the same way as every existing DAA scheme does. So in Table 1, we add this computation $n \cdot \mathbb{G}_T$. Another observation of this scheme is that the pairing computation in the Join protocol can be done

Table 1. Cost Comparison of the Six Pairing-Based DAA Protocols

Operation	Party	Computational Cost	Credential Size	Signature Size
Scheme of [13]				
Join	TPM	$3\mathbb{G}_1$	$3\mathbb{G}_1$	$2p + 3\mathbb{G}_1 + 2\mathsf{G} + 1h$
	Issuer	$2\mathbb{G}_1 + 2\mathbb{G}_1^2$		
	Host	$6P$		
Sign	TPM	$3\mathbb{G}_T$		
	Host	$3\mathbb{G}_1 + 1\mathbb{G}_T + 3P$		
Verify	Verifier	$1\mathbb{G}_T^2 + 1\mathbb{G}_T^3 + 5P + (n+1)\mathbb{G}_T$		
Scheme of [21]				
Join	TPM	$3\mathbb{G}_1$	$3\mathbb{G}_1$	$1p + 4\mathbb{G}_1 + 1h$
	Issuer	$2\mathbb{G}_1 + 2\mathbb{G}_1^2$		
	Host	$6P$		
Sign	TPM	$1\mathbb{G}_1$		
	Host	$4\mathbb{G}_1 + 3\mathbb{G}_T + 1P$		
Verify	Verifier	$1\mathbb{G}_1^2 + 1\mathbb{G}_T^2 + 5P + n\mathbb{G}_1$		
Scheme of [23]				
Join	TPM	$3\mathbb{G}_1$	$3\mathbb{G}_1$	$1p + 5\mathbb{G}_1 + 1h$
	Issuer	$2\mathbb{G}_1 + 2\mathbb{G}_1^2$		
	Host	$4P$		
Sign	TPM	$2\mathbb{G}_1 + 1\mathbb{G}_T$		
	Host	$3\mathbb{G}_1 + 1P$		
Verify	Verifier	$1\mathbb{G}_1^2 + 1\mathbb{G}_T^2 + 5P + n\mathbb{G}_1$		
Scheme of [24]				
Join	TPM	$3\mathbb{G}_1^2 + (2P)$	$2q + 1\mathbb{G}_1$	$6p + 2\mathbb{G}_1 + 2\mathsf{G} + 1h$
	Issuer	$1\mathbb{G}_1^2 + 1\mathbb{G}_1^3$		
	Host	$(2P)$		
Sign	TPM	$2\mathbb{G}_1 + 1\mathbb{G}_T^2$		
	Host	$1\mathbb{G}_1 + 2\mathbb{G}_1^2 + 1\mathbb{G}_1^3 + 1\mathbb{G}_T^3$		
Verify	Verifier	$1\mathbb{G}_1^2 + 2\mathbb{G}_1^3 + 1\mathbb{G}_T^5 + 3P + n\mathbb{G}_T$		
Scheme of [16]				
Join	TPM/Host	$2\mathbb{G}_1^2 + 1\mathbb{G}_2 + 2P + proof$	$2q + 1\mathbb{G}_1$	$4p + 1\mathbb{G}_1 + 2\mathsf{G} + 1h$
	Issuer	$1\mathbb{G}_1^2 + verify$		
Sign	TPM/Host	$1\mathbb{G}_1 + 2\mathbb{G}_T + 1\mathbb{G}_T^4$		
Verify	Verifier	$1\mathbb{G}_T^2 + 1\mathbb{G}_T^5 + 2P + n\mathbb{G}_T$		
Scheme of this paper				
Join	TPM	$2\mathbb{G}_1$	$1q + 1\mathbb{G}_1$	$4p + 3\mathbb{G}_1 + 1h$
	Issuer	$1\mathbb{G}_1 + 1\mathbb{G}_1^2$		
	Host	$1\mathbb{G}_1 + 2P$		
Sign	TPM	$2\mathbb{G}_1 + 1\mathbb{G}_T$		
	Host	$1\mathbb{G}_1 + 1\mathbb{G}_T^3$		
Verify	Verifier	$1\mathbb{G}_1^2 + 1\mathbb{G}_2^2 + 1\mathbb{G}_T^4 + 1P + n\mathbb{G}_1$		

by the Host instead of the TPM, because it is expensive to implement the pairing operation in TPMs. We mark this change as $(2P)$ in Table 1. Observe also that in the scheme in [16], the signer is a single entity, but it is not difficult to split the signer role between a TPM and a Host. So in the following comparison, we assume that these changes have been taken into account.

When a DAA scheme is used in the trusted computing environment, as the original design in [11], the most significant performance is a TPM's computational cost and storage requirement. As shown in the table, our proposed DAA scheme has the most efficient computational cost in the Join protocol, that includes not only the computational cost of the TPM but also the computational cost of the whole signer (the TPM and Host) and the computational cost of the issuer.

In the Sign protocol, the most efficient scheme regarding the TPM's operation is the scheme in [21]. However, this scheme is not secure as the attacks demonstrated in [20, 23]. Especially in the important operation of signing by the TPM, our scheme is the same as the scheme in [23], which is a repair of the scheme in [21] and these two schemes are more efficient than the other three schemes [13, 16, 24]. Note that in the scheme in [23], the pairing computation of the Host in the Sign protocol can be replaced by an exponentiation in $\mathbb{G}_T$ with the precompution on $\hat{t}(B, X)$. In the Verification algorithm, our scheme has the same computation cost of the verifier as in [16], and they are more efficient than all of the other schemes in [13, 21, 23, 24].

Except the efficiency in the computational cost, the attractive performance of our scheme is that it has the smallest credential size and signature size, comparing with the other schemes in the table. This comparison is based on the fact that the size of G is various, and to our best knowledge it can be chosen between the sizes of $\mathbb{G}_1$ and $\mathbb{G}_T$. In summary, our scheme is the most efficient DAA scheme so far with the acceptable security level.

6 Conclusions

In this paper[2], we have introduced a new DAA scheme, which is based on elliptic curves and asymmetric pairings. This scheme is more efficient than all the existing DAA schemes, in particular, this scheme requires very few TPM resources. We have also modified the definition of the user-controlled-traceability in [13] in order to cover the non-frameability property.

Recently TCG TPM working group has been working on the next generation of TPM. One of the most interesting features is that the new TPM supports algorithm agility. If a TPM supports multiple DAA algorithms, such as both RSA-DAA and ECC-DAA, we can make use of DAAseed as a master secret and create multiple DAA secret f values from it. In that case, we do not require extra

[2] This is a full version of the paper. An extended abstract of this paper appears in the Proceedings of the 5th China International Conference on Information Security and Cryptology (Inscrypt'09).

internal storage for long-term secrets. Our conclusion is that DAA is algorithm agile and our ECC-DAA implementation is fairly efficient.

Acknowledgements

The author would like to thank Jiangtao Li for his invaluable discussion and comments on this work, and also to thank the anonymous reviewers of Inscrypt'09 for their thoughtful comments, which are very helpful to improve the paper.

References

1. M. H. Au, W. Susilo and Y. Mu. Constant-size dynamic k-TAA. In *the Proceedings of 5th International Conference on Security and Cryptography for Networks – SCN 2006*, Springer-Verlag LNCS 4116, 111–125, 2006.
2. M. Backes, M. Maffei and D. Unruh. Zero knowledge in the applied Pi–calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy – SSP'08*, 202–215, 2008.
3. S. Balfe, A. D. Lakhani and K. G. Paterson. Securing peer-to-peer networks using trusted computing. In Mitchell (ed.), *Chapter 10 of Trusted Computing*. IEEE London, 271–298, 2005.
4. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC 2005*, Springer-Verlag LNCS 3897, 319–331, 2006.
5. M. Bellare, D. Micciancio and B. Warinschi. Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology — EUROCRYPT '03*, Springer-Verlag LNCS 2656, 614–629, 2003.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *the 1st ACM Conference on Computer and Communications Security*, 62–73, ACM Press, 1993.
7. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *Topics in Cryptology – CT-RSA 2005*, Springer-Verlag LNCS 3376, 136–153, 2005.
8. D. Boneh and X. Boyen. Sort signatures without random oracles. In *Advances in Cryptology - EUROCRYPT 2004*, LNCS 3027, 56–73, Springer-Verlag, 2004.
9. D. Boneh, X. Boyen and H. Shacham. Short group signatures. In *Advances in Cryptology - CRYPTO 2004*, LNCS 3152, 41–55, Springer-Verlag, 2004.
10. C. Boyd and C. Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology - Asiacrypt 2000*, Springer-Verlag LNCS 1976, 58–71, 2000.
11. E. Brickell, J. Camenisch and L. Chen. Direct anonymous attestation. In *the 11th ACM Conference on Computer and Communications Security*. ACM Press, 132–145, 2004.
12. E. Brickell, J. Camenisch and L. Chen. Direct anonymous attestation in context. In Mitchell (ed.), *Chapter 5 of Trusted Computing*. IEEE London, 143–174, 2005.
13. E. Brickell, L. Chen and J. Li. Simplified security notions for direct anonymous attestation and a concrete scheme from pairings. *Int. Journal of Information Security*, **8**, 315–330, 2009.

14. E. Brickell, L. Chen and J. Li. A new direct anonymous attestation scheme from bilinear maps. In *Trusted Computing - Challenges and Applications – TRUST 2008*, Springer-Verlag LNCS 4968, 166–178, 2008.
15. E. Brickell and J. Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In *the 6th ACM Workshop on Privacy in the Electronic Society (WPES 2007)*. ACM Press, 21–30, 2007.
16. E. Brickell and J. Li. Enhanced privacy ID from bilinear pairing. *Cryptology ePrint Archive*. Report 2009/095, available at `http://eprint.iacr.org/2009/095`.
17. J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *Security in Communication Networks – SCN 2004*. Springer-Verlag LNCS 3352, 122–135, 2004.
18. S. Canard and J. Traore. List signature schemes and application to electronic voting. Presented in *International Workshop on Coding and Cryptography 2003*. See also the journal version of this paper by S. Canard, B. Schoenmakers, M. Stam and J. Traore, titled "List signature schemes" in *Discrete Applied Mathematics* 154(2): 189-201, 2006.
19. L. Chen, Z. Cheng and N.P. Smart. Identity-based key agreement protocols from pairings. *Int. Journal of Information Security*, **6**, 213–242, 2007.
20. L. Chen and J. Li. A note on the Chen-Morrissey-Smart Direct Anonymous Attestation scheme. Preprint.
21. L. Chen, P. Morrissey and N. P. Smart. Pairings in trusted computing. In *Pairings in Cryptography – Pairing 2008*, Springer-Verlag LNCS 5209, 1–17, 2008.
22. L. Chen, P. Morrissey and N. P. Smart. On proofs of security of DAA schemes. In *Provable Security – ProvSec 2008*, Springer-Verlag LNCS 5324, 167–175, 2008.
23. L. Chen, P. Morrissey and N. P. Smart. DAA: Fixing the pairing based protocols. *Cryptology ePrint Archive*. Report 2009/198, available at `http://eprint.iacr.org/2009/198`.
24. X. Chen and D. Feng. Direct anonymous attestation for next generation TPM. *Journal of Computers*, **3**(12), 43–50. 2008.
25. C. Delerablee and D. Pointcheval. Dynamic fully anonymous short group signatures. In *Progress in Cryptology – Vietcrypt 2006*, Springer-Verlag LNCS 4341, pp. 193–210, 2006 (see also http://www.di.ens.fr/users/pointche/Documents/Papers/2006_vietcrypt.pdf for a corrected version of this paper).
26. S. Galbraith, K. Paterson and N.P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, **156**, 3113–3121, 2008.
27. H. Ge and S.R. Tate. A Direct anonymous attestation scheme for embedded devices. In *Public Key Cryptography – PKC 2007*, Springer-Verlag LNCS 4450, 2007.
28. ISO/IEC 11889:2009 Information technology – Security techniques – Trusted Platform Module.
29. ISO/IEC 14888-3 Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms.
30. A. Leung and L. Chen and C. J. Mitchell. On a possible privacy flaw in direct anonymous attestation (DAA). In *Trusted Computing – Challenges and Applications – TRUST 2008*. LNCS 4968, 179–190, Springer-Verlag, 2008.
31. A. Leung and C. J. Mitchell. Ninja: Non-identity based, privacy preserving authentication for ubiquitous environments. In *Ubiquitous Computing*, Springer-Verlag LNCS 4717, 73–90, 2007.
32. C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology – Crypto'97*, Springer-Verlag LNCS 1294, 249–263, 1997.

33. A. Lysyanskaya, R. Rivest, A. Sahai and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography – SAC 1999*, Springer-Verlag LNCS 1758, 184–199, 1999.
34. A. Pashalidis and C. J. Mitchell. Single sign-on using TCG-conformant platforms. In Mitchell (ed.), *Chapter 6 of Trusted Computing*. IEEE London, 175–193, 2005.
35. C. Rudolph. Covert identity information in direct anonymous attestation (DAA). In *the 22nd IFIP TC-11 International Information Security Conference – SEC2007*, 2007.
36. C.P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology – Crypto '89*, Springer-Verlag LNCS 435, 239-252, 1990.
37. B. Smyth, L. Chen and M. Ryan. Direct Anonymous Attestation (DAA): Ensuring privacy with corrupt administrators. In *Security and Privacy in Ad hoc and Sensor Networks – ESAS 2007*, Springer-Verlag LNCS 4572, 218–231, 2007.
38. Trusted Computing Group. TCG TPM specification 1.2. Available at `http://www.trustedcomputinggroup.org`, 2003.