Simplified Security Notions of Direct Anonymous Attestation and a Concrete Scheme from Pairings*

Ernie Brickell
Intel Corporation
ernie.brickell@intel.com

Liqun Chen
HP Laboratories
liqun.chen@hp.com

Jiangtao Li
Intel Corporation
jiangtao.li@intel.com

March 10, 2008

Abstract

Direct Anonymous Attestation (DAA) is a cryptographic mechanism that enables remote authentication of a user while preserving privacy under the user's control. The DAA scheme developed by Brickell, Camenisch, and Chen has been adopted by the Trust Computing Group (TCG) for remote anonymous attestation of Trusted Platform Module (TPM), a small hardware device with limited storage space and communication capability. In this paper, we provide two contributions to DAA. We first introduce simplified security notions of DAA including the formal definitions of user controlled anonymity and traceability. We then propose a new DAA scheme from elliptic curve cryptography and bilinear maps. The lengths of private keys and signatures in our scheme are much shorter than the lengths in the original DAA scheme, with a similar level of security and computational complexity. Our scheme builds upon the Camenisch-Lysyanskaya signature scheme and is efficient and provably secure in the random oracle model under the LRSW (stands for Lysyanskaya, Rivest, Sahai and Wolf) assumption and the decisional Bilinear Diffie-Hellman assumption.

Keywords: direct anonymous attestation, trusted computing, user-controlled-anonymity, user-controlled-traceability, bilinear maps.

1 Introduction

The concept and a concrete scheme of Direct Anonymous Attestation (DAA) were first introduced by Brickell, Camenisch, and Chen [8] for remote anonymous authentication of a hardware module, called Trusted Platform Module (TPM). The DAA scheme was adopted by the Trusted Computing Group (TCG) [34], 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 standardized in the TCG TPM Specification Version 1.2 [33]. A historical perspective on the development of DAA was provided by the DAA authors in [9].

A DAA scheme involves three types of entities: a DAA issuer, DAA signers, and DAA verifiers. The issuer is in charge of verifying the legitimation of signers and of issuing a DAA credential to each signer. A DAA signer can prove membership to a verifier by signing a DAA signature. The verifier can verify the membership credential from the signature but he cannot learn the identity of the signer. DAA is targeted for implementation in the TPM which has limited storage space and

*The pairing-based DAA scheme in this paper will be presented at the conference on Trusted Computing (TRUST 2008), Villach, Austria, March 2008.

computation capability. For this purpose, the role of the DAA signer is spilt between a TPM and a host that has the TPM "built 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 most interesting feature of DAA is to provide differing 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. However, 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 industry and cryptographic researches. Pashalidis and Mitchell showed how to use DAA in a single sign-on application [28]. Balfe, Lakhani and Paterson utilized a DAA scheme to enforce the use of stable, platform-dependent pseudonyms and reduce pseudo-spoofing in peer-to-peer networks [2]. Leung and Mitchell made use of a DAA scheme to build a non-identity-based authentication scheme for a mobile ubiquitous environment [24]. Camenisch and Groth [11] found that the performance of the original DAA scheme can be improved by introducing randomization of the RSA-based Camenisch-Lysyanskaya signature (CL-RSA) [12]. Rudolph [30] pointed out that if the DAA issuer is allowed to use multiple public keys each for issuing one credential only, it can violate anonymity of DAA. It is obviously true. Like a group signature scheme, the anonymous property is relied on the reasonable large size of the signer group that is associated with a single group manager's public key. Smyth, Chen and Ryan discussed how to ensure privacy when using DAA with corrupt administrators [32]. Backes et al. [1] presented a mechanized analysis of the original DAA scheme. Ge and Tate [23] proposed a very interesting DAA scheme with efficient signing and verification implementation but inefficient joining implementation. The security of this scheme, as the same as the original DAA scheme, is based on the strong RSA assumption and the decisional Diffie-Hellman assumption.

In this paper, we provide two contributions to DAA. Our first contribution is the simplified security notions of DAA. We introduce a new interpretation of formal specification and security model of DAA, which is intended to address the same concept of a DAA scheme and to cover the same security properties that the DAA scheme should hold, as introduced in [8]. The new security model of a DAA scheme is specified with two new security notions: user-controlled-anonymity and user-controlled-traceability. We formally define the two notions, and also discuss the differentiation between a DAA scheme and a group signature scheme from a perspective on an adversary's behavior; in particular we compare the two notions with full-anonymity and full-traceability of group signatures, as defined in [4]. In the hope of the authors, this interpretation would be easier to read than the reletive content in [8], and would be helpful for readers to understand and accept the concept of DAA and its security requirements.

Our second contribution is a new concrete DAA scheme from elliptic curve cryptography and bilinear maps. This DAA scheme builds on top of the Camenisch and Lysyanskaya signature scheme [13] based on the LRSW assumption [26] (CL-LRSW). One limitation of the original DAA scheme [8] is that the lengths of private keys and DAA signatures are quite large for a small TPM, i.e., around 670 bytes and 2800 bytes, respectively. Our new DAA scheme requires a much shorter key length compared with the original integer factorization based DAA scheme. The lengths of private keys and signatures in our new scheme are approximately 213 bytes and 521 bytes, respectively, with a similar level of security and computational complexity. We prove that this DAA scheme is secure based on our new security notions in the random oracle model and under the LRSW assumption and the decisional Bilinear Diffie-Hellman assumption.

Rest of this paper is organized as follows. We first introduce the new security notions in Section 2. We then briefly review the notations on bilinear maps, some relative security assumptions, and some known cryptographic building blocks in Section 3, which will be used in the description of our new DAA scheme in Section 4 and the corresponding security proofs in Section 5. After that we show how to implement the new DAA scheme in Section 6, and conclude the paper in Section 7.

2 Simplified Security Notions of DAA

In this section, we present the new formal specification and security model of DAA. First of all, four players in a DAA scheme are denoted as follows: a DAA issuer $\mathcal{I}$, a TPM $\mathcal{M}_i$, a host $\mathcal{H}_i$ and a verifier $\mathcal{V}_j$. $\mathcal{M}_i$ and $\mathcal{H}_i$ form a platform in the trusted computing environment and share the role of a DAA signer. Regarding their security attributes, we consider the following three cases: (1) neither $\mathcal{M}_i$ nor $\mathcal{H}_i$ is corrupted by an adversary, (2) both of them are corrupted, and (3) $\mathcal{H}_i$ is corrupted but not $\mathcal{M}_i$. Like in [8], we do not consider the case that $\mathcal{M}_i$ is corrupted but not $\mathcal{H}_i$.

2.1 Specification of DAA

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

- Setup: On input of a security parameter 1^k, $\mathcal{I}$ uses this randomized algorithm to produce a pair ($\mathtt{isk}$, $\mathtt{params}$), where $\mathtt{isk}$ is the issuer's secret key, and $\mathtt{params}$ is the global public parameters for the system, including the issuer's public key $\mathtt{ipk}$, 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 $\mathtt{params}$ are publicly known so that we do not need to explicitly provide them as input to other algorithms.

- Join: This randomized algorithm has two sub-algorithms, namely $\mathsf{Join_t}$ and $\mathsf{Join_i}$. $\mathcal{M}_i$ uses $\mathsf{Join_t}$ to produce a pair $(\mathtt{tsk}_i, \mathtt{comm}_i)$, where $\mathtt{tsk}_i$ is the TPM's secret key and $\mathtt{comm}_i$ is a commitment of $\mathtt{tsk}_i$ associated with the issuer $\mathcal{I}$. On input of $\mathtt{comm}_i$ and $\mathtt{isk}$, $\mathcal{I}$ uses $\mathsf{Join_i}$ to produce $\mathtt{cre}_i$, which is a DAA credential associated with $\mathtt{tsk}_i$. Note that the value $\mathtt{cre}_i$ is given to both $\mathcal{M}_i$ and $\mathcal{H}_i$, but the value $\mathtt{tsk}_i$ is known to $\mathcal{M}_i$ only.

- Sign: On input of $\mathtt{tsk}_i$, $\mathtt{cre}_i$, a basename $\mathtt{bsn}_j$ (the name string of $\mathcal{V}_j$ 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, $\mathcal{M}_i$ and $\mathcal{H}_i$ use this randomized algorithm to produce a signature σ on m under $(\mathtt{tsk}_i, \mathtt{cre}_i)$ associated with $\mathtt{bsn}_j$. The basename $\mathtt{bsn}_j$ is used for controlling the linkability.

- Verify: On input of m, $\mathtt{bsn}_j$, a candidate signature σ for m, and a set of rogue signers' secret keys $\mathtt{ROGUE}$, $\mathcal{V}_j$ uses this deterministic algorithm to return either 1 (accept) or 0 (reject). Note also that how to build the set of $\mathtt{ROGUE}$ is out the scope of the DAA scheme.

- Link: On input of two signatures σ_0 and σ_1, $\mathcal{V}_j$ uses this deterministic algorithm to return 1 (linked), 0 (unlinked) or $\perp$ (invalid signatures). Link will output $\perp$ if, by using an empty $\mathtt{ROGUE}$ (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 $\mathtt{tsk} \in \mathtt{ROGUE}$ or not.

2.2 Security Notions

In our new security model of DAA, a DAA scheme must hold the notions of correctness, user-controlled-anonymity and user-controlled-traceability, as defined in this section.

2.2.1 Correctness

If both the signer and verifier are honest, that implies $\mathtt{tsk}_i \notin \mathtt{ROGUE}$, the signatures and their links generated by the signer will be accepted by the verifier with overwhelming probability. This means that the algorithms specified in Section 2.1 must meet the following consistency requirement. If

$$(\mathtt{isk}, \mathtt{params}) \leftarrow \mathsf{Setup}(1^k), \quad (\mathtt{tsk}_i, \mathtt{cre}_i) \leftarrow \mathsf{Join}(\mathtt{isk}, \mathtt{params}) \text{ and}$$

$$(m_b, \sigma_b) \leftarrow \mathsf{Sign}(m_b, \mathtt{bsn}_j, \mathtt{tsk}_i, \mathtt{cre}_i, \mathtt{params})|_{b=\{0,1\}},$$

then we must have

$$1 \leftarrow \mathsf{Verify}(m_b, \mathtt{bsn}_j, \sigma_b, \mathtt{params}, \mathtt{ROGUE})|_{b=\{0,1\}} \text{ and}$$

$$1 \leftarrow \mathsf{Link}(\sigma_0, \sigma_1, \mathtt{params})|_{\mathtt{bsn}_j \neq \perp}.$$

2.2.2 User-Controlled-Anonymity

Informally, the notion of user-controlled-anonymity requires that the following two properties are held in the DAA scheme:

1. *Anonymity.* An adversary not in possession of the signer's secret key finds it hard to recover the identity of the signer from its signature.

2. *User-controlled unlinkability.* Given two signatures σ_0 and σ_1 associated with two basenames $\mathtt{bsn}_0$ and $\mathtt{bsn}_1$ respectively, where $\mathtt{bsn}_0 \neq \mathtt{bsn}_1$, an adversary not in possession of the secret key(s) of the signer(s) finds it hard to tell whether or not the two signatures are signed by the same signer.

The notion of user-controlled-anonymity is different from the notion of anonymity and unlinkability in a group signature scheme, such as namely full-anonymity defined in [4], in the following two aspects. First, in the definition of full-anonymity, the adversary does not possess the group manager's secret key, but is allowed to corrupt all group members, including the signer itself. In the definition of user-controlled-anonymity, the adversary is allowed to corrupt the group manager (namely the DAA issuer $\mathcal{I}$) and a number of group members (namely the TPM $\mathcal{M}_i$ and the host $\mathcal{H}_i$), but not all group members. Actually, in out formal definition below, at least two group members, including the signer, must not be corrupted. Secondly, in the definition of full-anonymity, the adversary not in possession of the group manager's secret key cannot link any two given signatures w.r.t. the identities of their signers. But in user-controlled-anonymity, a signer, by negotiating with a verifier, can decide whether or not to let the verifier know the link.

To formalize the process, we define this notion via a game played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. In this game, $\mathcal{A}$ is allowed to corrupt $\mathcal{I}$ and to obtain all signers' credentials; $\mathcal{A}$ is also allowed to create and to corrupt a polynomial number of signers. $\mathcal{A}$ can freely choose the identities of the signers that are created/corrupted and the number of the created/corrupted signers.

Game of User-Controlled-Anonymity

- *Initial:* $\mathcal{C}$ runs $\mathsf{Setup}(1^k)$ and gives the resulting $\mathtt{isk}$ and $\mathtt{params}$ to $\mathcal{A}$. Alternatively, $\mathcal{C}$ receives the values $\mathtt{isk}$ and $\mathtt{params}$ from $\mathcal{A}$ with a request for initiating the game, and then verifies the validation of ($\mathtt{isk}$, $\mathtt{params}$).

- *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 $\texttt{bsn}$ (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 $\mathsf{Join_t}$ with $\mathcal{A}$ to create $\texttt{tsk}$ and to obtain $\texttt{cre}$ from $\mathcal{A}$. $\mathcal{C}$ verifies the validation of $\texttt{cre}$ and keeps $\texttt{tsk}$ secret.

– Corrupt. $\mathcal{A}$ submits a signer's identity ID of his choice to $\mathcal{C}$, who responds with the value $\texttt{tsk}$ 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 $\texttt{bsn}$ 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 $\texttt{bsn}$ if $\texttt{bsn} \neq \perp$ with either ID_0 or ID_1. To make the challenge, $\mathcal{C}$ chooses a bit b uniformly at random, signs m associated with $\texttt{bsn}$ under $(\texttt{tsk}_b, \texttt{cre}_b)$ 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 $\texttt{bsn}$ if $\texttt{bsn} \neq \perp$ 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.*

2.2.3 User-Controlled-Traceability

Informally, the notion of user-controlled-Traceability requires that the following two properties are held in the DAA scheme:

1. *Unforgeability.* An adversary, which has corrupted a set of signers' secret keys and their credentials, finds it hard to forge a valid signature under a secret key and credential, which is not in the set.

2. *User-controlled linkability.* Given a single basename $\texttt{bsn}$ $(\neq \perp)$, an adversary finds it hard to create two different signatures under the same $\texttt{tsk}_i$ and both of the signatures are associated with $\texttt{bsn}$, but the output of the algorithm Link is 0 (unlinked).

The notion of user-controlled-traceability is different from the notion of traceability in a group signature scheme, such as namely full-traceability defined in [4]. In the definition of full-traceability, there is an open algorithm, which allows the group manager to open the identity of any signer from its signature. In the definition of user-controlled-traceability, there is a link algorithm rather than the open algorithm, which allows a verifier to recognize whether two signatures are linked or not. However, the link algorithm only works under the signer's control. If the signer is willing to offer such a link between his two signatures, he makes use of the single $\texttt{bsn}$ $(\neq \perp)$ in the two signing processes; otherwise he will make use of different $\texttt{bsn}$ values or $\texttt{bsn} = \perp$. In that case, no linkage can be detected by anybody; it therefore offers zero traceability. In the other hand, if a verifier is unhappy with the zero traceability, he can insist that he should only accept the DAA signatures associated with the $\texttt{bsn}$ of his choice. In order to meet the conflict requirements between the signer and verifier, the DAA scheme provides a room for the signer and verifier to negotiate this matter; as a result, a good balance between security and privacy is achieved with DAA.

Like the notion of full-traceability, the notion of user-controlled-traceability covers the property of unforgeability (following the first item above), and the property of collusion resistance, which means that no adversary can output a new signer's secret key, given a set of the existing corrupted signer secret keys. Obviously this property holds by following the first item above as well, since anyone with the knowledge of a new signer secret key can easily forge a new signature. As mentioned before, the DAA scheme allows to separate the role of the signer between the TPM $\mathcal{M}_i$ and host $\mathcal{H}_i$, where $\mathcal{M}_i$ holds $(\mathtt{tsk}_i, \mathtt{cre}_i)$ and $\mathcal{H}_i$ knows $\mathtt{cre}_i$ only. By using a special Semi-sign query defined below, the notion of user-controlled-traceability also covers the property of security against a malicious host, that means $\mathcal{H}_i$ using $\mathtt{cre}_i$ cannot create a valid signature on behalf of $\mathcal{M}_i$.

To formalize the process, we define this notion via a game played by a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, in which $\mathcal{A}$ is not allowed to corrupt $\mathcal{I}$, but $\mathcal{A}$ is allowed to create and to corrupt a polynomial number of signers. The identities of the signers that are corrupted and their number are entirely up to $\mathcal{A}$.

Game of User-Controlled-Traceability

- *Initial:* $\mathcal{C}$ runs $\mathsf{Setup}(1^k)$ and gives the resulting `params` to $\mathcal{A}$. It keeps `isk` secret.
- *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 $\mathcal{H}_i$ to $\mathcal{M}_i$ in Sign of his choice to $\mathcal{C}$, who acts as $\mathcal{M}_i$ in Sign and responds with the data transmitted from $\mathcal{M}_i$ to $\mathcal{H}_i$ in Sign.
  - Join. There are two cases of this query. Case 1: $\mathcal{A}$ submits a signer's identity ID of his choice to $\mathcal{C}$, who runs Join to create `tsk` and `cre` for the signer. Case 2: $\mathcal{A}$ submits a signer's identity ID with a `tsk` value of his choice to $\mathcal{C}$, who runs $\mathsf{Join_I}$ to create `cre` for the signer and puts the given `tsk` into the list of `ROGUE`. $\mathcal{C}$ responds the query with `cre`. Suppose that $\mathcal{A}$ does not use a single ID for both of the cases.
  - Corrupt. The same as in the game of user-controlled-anonymity, except that at the end $\mathcal{C}$ puts the revealed `tsk` into the list of `ROGUE`.
- *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
  1. $\mathsf{Verify}(m, \mathtt{bsn}, \sigma, \mathtt{ROGUE}) = 1$ (accepted), but σ is neither a response of the existing Sign queries nor a response of the existing Semi-sign queries (partially); and/or
  2. In the case of $\mathtt{bsn} \neq \perp$, there exists another signature σ' associated with the same identity and `bsn`, and the output of $\mathsf{Link}(\sigma, \sigma')$ is 0 (unlinked).

Note that in the first item, the fact Verify outputs 1 implies that the corresponding $\mathtt{cre} \notin \mathtt{ROGUE}$, i.e. the `tsk` is neither through Case 2 of the Join query nor from the Corrupted query. This item means that $\mathcal{A}$ is able to forge a DAA signature without knowing `tsk` though it may know `cre`.

Note also that in the second item, the linkage between σ and σ' that are associated with the same `tsk` and `bsn` ($\neq \perp$) is not detected. As mentioned in the definition of Link, the output 0 implies that $\mathsf{Verify}(\sigma) = 1$ using an empty `ROGUE`. This item means that $\mathcal{A}$ is able to create a signature which can escape from the signer and verifier pre-agreed linkability. Therefore this situation covers an insider attack from a legitimate signer, who knows `tsk`.

Definition 2 *Let $\mathcal{A}$ denote an adversary that plays the game above. We denote by* $\mathbf{Adv}[\mathcal{A}^{trace}_{\mathcal{DAA}}] = \mathbf{Pr}[\mathcal{A}\ wins]$ *the advantage of $\mathcal{A}$ in breaking 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.*

3 Background and Building Blocks

3.1 Background on Bilinear Maps

As in Boneh and Franklin's identity-based encryption scheme [7] and Camenisch and Lysyanskaya (CL-LRSW) signature scheme [13], our new DAA scheme makes use of a bilinear map $e : G \times G \to \mathsf{G}$, where G and G denotes two groups of prime order q. The map e satisfies the following properties:

1. Bilinear. For all $P, Q \in G$, and for all $a, b \in \mathbb{Z}_q$, $e(P^a, Q^b) = e(P, Q)^{ab}$.
2. Non-degenerate. There exists some $P, Q \in G$ such that $e(P, Q)$ is not the identity of G.
3. Computable. There exists an efficient algorithm for computing $e(P, Q)$ for any $P, Q \in G$.

A bilinear map satisfying the above properties is said to be an admissible bilinear map. Such bilinear map is also known as the symmetric pairing. In Section 6, we will give a concrete example of groups G, G and an admissible bilinear map between them. The group G is a subgroup of the group of points of an elliptic curve $E(\mathbb{F}_p)$ for a large prime p. The group G is a subgroup of the multiplicative group of a finite field $\mathbb{F}^*_{p^2}$. We can use the Tate pairing to construct an admissible bilinear map between these two groups.

In general, one can consider bilinear maps $e : G_1 \times G_2 \to \mathsf{G}$ where G_1, G_2, G are cyclic groups of prime order q. However in our paper, we limit ourself to symmetric pairing where $G_1 = G_2$, because our scheme builds upon the CL-LRSW signature scheme, which uses the symmetric pairing.

3.2 Cryptographic Assumptions

The security of our DAA scheme relies on the Decisional Bilinear Diffie-Hellman (DBDH) assumption, and the Lysyanskaya, Rivest, Sahai, and Wolf (LRSW) assumption. We now state these assumptions as follows:

Assumption 1 (DBDH Assumption) *Let $G = \langle g \rangle$ be a bilinear group defined above of prime order q. For sufficiently large q, the distribution $\{(g, g^a, g^b, g^c, e(g, g)^{abc})\}$ is computationally indistinguishable from the distribution $\{(g, g^a, g^b, g^c, e(g, g)^d)\}$, where a, b, c, and d are random elements in $\mathbb{Z}_q$.*

The DBDH assumption is a natural combination of the Decisional Diffie-Hellman assumption and Bilinear Diffie-Hellman assumption. It has been used in many cryptographic schemes, e.g., Boneh and Boyen's construction of secure identity based encryption scheme without random oracles [6].

Assumption 2 (LRSW Assumption) *Let $G = \langle g \rangle$ be a cyclic group, $X, Y \in G$, $X = g^x$, and $Y = g^y$. Suppose there is an oracle that, on input $m \in \mathbb{Z}_q$, outputs a triple (a, a^y, a^{x+mxy}) for a randomly chosen $a \in G$. Then there exists no efficient adversary that queries the oracle polynomial number of times, and outputs (m, a, b, c) such that $m \neq 0$, $b = a^y$ and $c = a^{x+mxy}$ where m has not been queried before.*

The LRSW assumption was introduced by Lysyanskaya et al. [26] and was shown that this assumption holds for generic groups. This assumption is also used in the CL-LRSW signature scheme [13].

3.3 Protocols for Proof of Knowledge

In our scheme we will use various protocols to prove knowledge of and relations among discrete logarithms. To describe these protocols, we use notation introduced by Camenisch and Stadler [15] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For example, $PK\{(a,b) : y_1 = g_1^a h_1^b \wedge y_2 = g_2^a h_2^b\}$ denotes a proof of knowledge of integers a and b such that $y_1 = g_1^a h_1^b$ and $y_2 = g_2^a h_2^b$ holds, where $y_1, g_1, h_1, y_2, g_2, h_2$ are elements of some groups $G_1 = \langle g_1 \rangle = \langle h_1 \rangle$ and $G_2 = \langle g_2 \rangle = \langle h_2 \rangle$. The variables in the parenthesis denote the values the knowledge of which is being proved, while all other parameters are known to the verifier. Using this notation, a proof of knowledge protocol can be described without getting into all details. In the random oracle model, such proof of knowledge protocols can be turned into signature schemes using the Fiat-Shamir heuristic [20, 29]. We use the notation $SPK\{(a) : y = z^a\}(m)$ to denote a signature on a message m obtained in this way.

In this paper, we use the following known proof of knowledge protocols:

- Proof of knowledge of discrete logarithms. A proof of knowledge of a discrete logarithm of an element $y \in G$ with respect to a base z is denoted as $PK\{(a) : y = z^a\}$. The discrete logarithms in such proof of knowledge protocol can be modulo a prime [31] or a composite [19, 21], where the composite is a safe-prime product. A proof of knowledge of a representation of an element $y \in G$ with respect to several bases $z_1, \ldots, z_v \in G$ [17] is denoted $PK\{(a_1, \ldots, a_v) : y = z_1^{a_1} \cdot \ldots \cdot z_v^{a_v}\}$.
- Proof of knowledge of equality. A proof of equality of discrete logarithms of two group elements $y_1, y_2 \in G$ to the bases $z_1, z_2 \in G$, respectively, [16, 18] is denoted $PK\{(a) : y_1 = z_1^a \wedge y_2 = z_2^a\}$. Such protocol can also be used to prove that the discrete logarithms of two group elements $y_1 \in G_1$ and $y_2 \in G_2$ to the bases $z_1 \in G_1$ and $z_2 \in G_2$, respectively, in two different groups G_1 and G_2 are equal [10, 14].

3.4 Camenisch-Lysyanskaya Signature Scheme

Our DAA scheme is based on the Camenisch-Lysyanskaya (CL-LRSW) signature scheme [13]. Unlike most signature schemes, this one is particularly suited for our purposes as (1) it uses bilinear maps and (2) it allows for efficient protocols to prove knowledge of a signature and to obtain a signature on a secret message based on proofs of knowledge. We now review the CL-LRSW signature scheme as follows.

Key Generation. Chooose two groups $G = \langle g \rangle$ and $\mathsf{G} = \langle \mathsf{g} \rangle$ of prime order q and an admissible bilinear map e between G and G. Next choose $x \leftarrow \mathbb{Z}_q$ and $y \leftarrow \mathbb{Z}_q$, and set the public key as $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ and the secret key as (x, y), where $X = g^x$ and $Y = g^y$.

Signature. On input a message m, the secret key (x, y), and the public key $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, choose a random $a \in G$, and output the signature $\sigma = (a, a^y, a^{x+mxy})$.

Verification. On input the public key $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, the message m, and the signature $\sigma = (a, b, c)$ on m, check whether the following equations hold $e(Y, a) \stackrel{?}{=} e(g, b)$, $e(X, a) \cdot e(X, b)^m \stackrel{?}{=} e(g, c)$.

Observe that from a signature $\sigma = (a, b, c)$ on a message m, it is easy to compute a different signature $\sigma' = (a', b', c')$ on the same message m without the knowledge of the secret key: just choose a random number $r \in \mathbb{Z}_q$ and compute $a' := a^r$, $b' := b^r$, $c' := c^r$.

Theorem 1 ([13]) *The CL-LRSW signature scheme is secure against adaptive chosen message attacks under the LRSW assumption.*

4 The New DAA Scheme from Bilinear Maps

We now present a new DAA scheme from bilinear maps based on the CL-LRSW signature scheme [13]. In the DAA scheme, there are three types of entities: an issuer $\mathcal{I}$, signers, and verifiers $\mathcal{V}$. Each signer (a trusted platform) consists of a host $\mathcal{H}$ and a TPM $\mathcal{M}$. All communications between $\mathcal{M}$ and $\mathcal{I}$ are through $\mathcal{H}$. For any computation performed by a signer, part of it is performed on the $\mathcal{M}$ while the rest of the computation is done on $\mathcal{H}$. Since the TPM is a small chip with limited resources, a requirement for DAA is that the operations carried out on the TPM should be minimal. We have the following five operations:

Setup Let ℓ_q, ℓ_H, and ℓ_ϕ be three security parameters, where ℓ_q is the size of the order q of the groups, ℓ_H is the output length of the hash function used for Fiat-Shamir heuristic, and ℓ_ϕ is the security parameter controlling the statistical zero-knowledge property. The issuer $\mathcal{I}$ chooses two groups $G = \langle g \rangle$ and $\mathsf{G} = \langle \mathsf{g} \rangle$ of prime order q and an admissible bilinear map e between G and G, i.e., $e : G \times G \to \mathsf{G}$. $\mathcal{I}$ then chooses $x \leftarrow \mathbb{Z}_q$ and $y \leftarrow \mathbb{Z}_q$ uniformly at random, and computes $X := g^x$ and $Y := g^y$. For simplicity, throughout the scheme specification, the exponentiation operation h^a for $h \in G$ and an integer a outputs an element in G. $\mathcal{I}$ sets the group public key as $\mathtt{ipk} := (q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ and its private key as $\mathtt{isk} := (x, y)$, and publishes $\mathtt{ipk}$. Let $H(\cdot)$ and $H_{\mathsf{G}}(\cdot)$ be two collision resistant hash functions, such that $H : \{0,1\}^* \to \{0,1\}^{\ell_H}$ and $H_{\mathsf{G}} : \{0,1\}^* \to \mathsf{G}$. Observe that the correctness of the group public key can be verified, e.g. by the host $\mathcal{H}$, by checking whether each element is in the right groups or not.

Join We assume that the signer and the issuer $\mathcal{I}$ have established a one-way authentic channel, i.e., $\mathcal{I}$ needs to be sure that it talks to the right signer (i.e. a platform with a specific TPM). The authentic channel can be achieved in various ways. The one TCG recommended is that every message sent from $\mathcal{I}$ to the signer is encrypted under the TPM endorsement key [33]. Let $\mathtt{ipk} = (q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, K_I be a long-term public key of $\mathcal{I}$. Let $\mathtt{DAAseed}$ be the seed to compute the secret key $\mathtt{tsk}$ of the signer. Note that we do not include the issuer's basename $\mathtt{bsn}_I$ in this operation, since we assume that the value g is unique for the issuer. However, if a different $\mathtt{bsn}_I$ is required, it can be added easily in the same way as in the original DAA scheme [8]. The join protocol takes the following steps:

1. $\mathcal{M}$ first computes
$$f := H(\mathtt{DAAseed} \| K_I) \bmod q, \qquad\qquad F := g^f,$$
where $\|$ stands for the operation of concatenation. $\mathcal{M}$ then chooses a random $r_f \leftarrow \mathbb{Z}_q$, computes $\tilde{T} := g^{r_f}$ and sends $\tilde{T}$ and F to $\mathcal{H}$.
2. The issuer chooses a random string $n_I \in \{0,1\}^{\ell_H}$ and sends it to $\mathcal{H}$.
3. $\mathcal{H}$ computes $\mathfrak{c}_h := H(q \| g \| \mathsf{g} \| X \| Y \| F \| \tilde{T} \| n_I)$ and sends it to $\mathcal{M}$.

4. $\mathcal{M}$ chooses a random string $n_T \in \{0,1\}^{\ell_\phi}$ and computes $\mathfrak{c} := H(\mathfrak{c}_h \| n_T)$ and $s_f := r_f + \mathfrak{c} \cdot f \bmod q$. $\mathcal{M}$ sets f as its `tsk` and sends $(F, \mathfrak{c}, s_f, n_T)$ to $\mathcal{I}$ through $\mathcal{H}$.
5. $\mathcal{I}$ checks its record and policy to find out whether the value F should be rejected or not. If F belongs to a rogue TPM or does not pass the issuer's policy check, e.g. having been required for an credential too many times, $\mathcal{I}$ aborts the protocol.
6. $\mathcal{I}$ computes $\hat{T} := g^{s_f} F^{-\mathfrak{c}}$, and verifies that $\mathfrak{c} \stackrel{?}{=} H(H(q\|g\|\mathsf{g}\|X\|Y\|F\|\hat{T}\|n_I)\|n_T)$. If the verification fails, $\mathcal{I}$ aborts. Otherwise, $\mathcal{I}$ chooses $r \leftarrow \mathbb{Z}_q$, and computes

$$a := g^r, \qquad b := a^y, \qquad c := a^x F^{rxy}.$$

Observe that $c = a^x g^{frxy} = a^{x+fxy}$ and therefore (a, b, c) is a CL-LRSW signature on f. $\mathcal{I}$ sets $\texttt{cre} = (a, b, c)$ to be the credential for $\mathcal{M}$, and sends `cre` to the signer, $\mathcal{M}$ and $\mathcal{H}$.
7. If verifying `cre` is required, $\mathcal{M}$ computes $d = b^f$ and sends it to $\mathcal{H}$, who then verifies whether $e(Y, a) = e(g, b)$, $e(g, d) = e(F, b)$, and $e(X, ad) = e(g, c)$ hold.

Sign Let m be the message to be signed (as the same as in the original DAA scheme, m is presented as $b\|m'$ where $\mathrm{b} = 0$ means that the message m' is generated by the TPM and $\mathrm{b} = 1$ means that m' was input to the TPM), $\texttt{bsn}_V$ be a basename associated with $\mathcal{V}$, and $n_V \in \{0,1\}^{\ell_H}$ be a nonce provided by the verifier. $\mathcal{M}$ has a secret key $\texttt{tsk} = f$ and a credential $\texttt{cre} = (a, b, c)$, whereas $\mathcal{H}$ only knows the credential `cre`. The signing algorithm takes the following steps:

1. Depending on whether $\texttt{bsn}_V = \perp$ or not, $\mathcal{H}$ computes B as follows

$$\mathsf{B} \stackrel{R}{\leftarrow} \mathsf{G} \qquad \text{or} \qquad \mathsf{B} := H_{\mathsf{G}}(1\|\texttt{bsn}_V),$$

where $\mathsf{B} \stackrel{R}{\leftarrow} \mathsf{G}$ means that B is chosen from G uniformly at random. $\mathcal{H}$ sends B to $\mathcal{M}$.
2. $\mathcal{M}$ verifies that $\mathsf{B} \in \mathsf{G}$ then computes $\mathsf{K} := \mathsf{B}^f$, and sends K to $\mathcal{H}$.
3. $\mathcal{H}$ chooses two integers $r, r' \leftarrow \mathbb{Z}_q$ uniformly at random and computes

$$a' := a^{r'}, \qquad b' := b^{r'}, \qquad c' := c^{r' r^{-1}},$$
$$\mathsf{v}_x := e(X, a'), \qquad \mathsf{v}_{xy} := e(X, b'), \qquad \mathsf{v}_s := e(g, c').$$

4. $\mathcal{H}$ sends v_{xy} back to $\mathcal{M}$ who later verifies that $\mathsf{v}_{xy} \in \mathsf{G}$.
5. $\mathcal{M}$ and $\mathcal{H}$ jointly compute a "signature proof of knowledge" as follows

$$SPK\{(r, f) : \mathsf{v}_s^r = \mathsf{v}_x \mathsf{v}_{xy}^f \ \wedge \ \mathsf{K} = \mathsf{B}^f\}(n_V, n_T, m).$$

   (a) $\mathcal{H}$ chooses a random integer $r_r \in \mathbb{Z}_q$ and computes $\tilde{\mathsf{T}}_{1t} := \mathsf{v}_s^{r_r}$.
   (b) $\mathcal{H}$ computes $\mathfrak{c}_H := H(q\|g\|\mathsf{g}\|X\|Y\|a'\|b'\|c'\|\mathsf{v}_x\|\mathsf{v}_{xy}\|\mathsf{v}_s\|\mathsf{B}\|\mathsf{K}\|n_V)$ and sends $\mathfrak{c}_H$ and $\tilde{\mathsf{T}}_{1t}$ to $\mathcal{M}$.
   (c) $\mathcal{M}$ chooses a random integer $r_f \leftarrow \mathbb{Z}_q$ and a nonce $n_T \in \{0,1\}^{\ell_\phi}$ and computes

$$\tilde{\mathsf{T}}_1 := \tilde{\mathsf{T}}_{1t} \mathsf{v}_{xy}^{-r_f}, \quad \tilde{\mathsf{T}}_2 := \mathsf{B}^{r_f}, \quad \mathfrak{c} := H(\mathfrak{c}_H \| \tilde{\mathsf{T}}_1 \| \tilde{\mathsf{T}}_2 \| n_T \| m), \quad s_f := r_f + \mathfrak{c} \cdot f \bmod q.$$

   (d) $\mathcal{M}$ sends $\mathfrak{c}$, s_f and n_T to $\mathcal{H}$.
   (e) $\mathcal{H}$ computes $s_r := r_r + \mathfrak{c} \cdot r \bmod q$.
6. $\mathcal{H}$ outputs the signature $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$ along with n_T.

Verify The input to this program is the group public key $\texttt{ipk} = (q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, a message m, two nonces n_V and n_T, the basename $\texttt{bsn}_V$, a candidate signature $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$ on (m, n_V, n_T), and a list of rogue secrete keys $\texttt{ROGUE}$, the verifier $\mathcal{V}$ does the following:

1. If $\texttt{bsn}_V \neq \perp$, $\mathcal{V}$ verifies that $\mathsf{B} \stackrel{?}{=} H_{\mathsf{G}}(1\|\texttt{bsn}_V)$, otherwise, $\mathcal{V}$ verifies that $\mathsf{B} \stackrel{?}{\in} \mathsf{G}$.
2. For each f_i in $\texttt{ROGUE}$, $\mathcal{V}$ checks that $\mathsf{K} \stackrel{?}{\neq} \mathsf{B}^{f_i}$. If K matches with any f_i in $\texttt{ROGUE}$, $\mathcal{V}$ outputs 0 (reject) and aborts.
3. $\mathcal{V}$ verifies that $e(a', Y) \stackrel{?}{=} e(g, b')$ and $\mathsf{K} \stackrel{?}{\in} \mathsf{G}$.
4. $\mathcal{V}$ computes
$$\begin{array}{lll} \hat{\mathsf{v}}_x := e(X, a'), & \hat{\mathsf{v}}_{xy} := e(X, b'), & \hat{\mathsf{v}}_s := e(g, c'), \\ \hat{\mathsf{T}}_1 := \hat{\mathsf{v}}_s^{s_r} \hat{\mathsf{v}}_{xy}^{-s_f} \hat{\mathsf{v}}_x^{-\mathfrak{c}}, & \hat{\mathsf{T}}_2 := \mathsf{B}^{s_f} \mathsf{K}^{-\mathfrak{c}}. & \end{array}$$
5. $\mathcal{V}$ verifies that $\mathfrak{c} \stackrel{?}{=} H(H(q\|g\|\mathsf{g}\|X\|Y\|a'\|b'\|c'\|\hat{\mathsf{v}}_x\|\hat{\mathsf{v}}_{xy}\|\hat{\mathsf{v}}_s\|\mathsf{B}\|\mathsf{K}\|n_V)\|\hat{\mathsf{T}}_1\|\hat{\mathsf{T}}_2\|n_T\|m)$.
6. If all the above verifications succeed, $\mathcal{V}$ outputs 1 (accept), otherwise $\mathcal{V}$ outputs 0 (reject).

Link When the verifier $\mathcal{V}$ wants to check whether or not two given signatures σ and σ' are linked, i.e. signed under the same $\texttt{tsk}$, it first checks the validation of them by using the above Verify algorithm but ignores the rogue TPM check in Item 2. If either of them is invalid, $\mathcal{V}$ outputs $\perp$. Otherwise, $\mathcal{V}$ outputs 1 (linked) if σ and σ' include the same (B, K) pair or 0 (unlinked) otherwise.

5 Security Results

In this section, we will state the security results for the new DAA scheme specified in Section 4 under the definitions of security notions in Section 2.2. 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 DBDH assumption as defined in Assumption 1 and the LRSW assumption as defined in Assumption 2 of Section 3. The security analysis of the notions of user-controlled-anonymity and user-controlled-traceability is in the random oracle model [5], i.e., we will assume that the hash functions H and H_{G} in the new DAA scheme are random oracles.

Theorem 2 *The DAA scheme specified in Section 4 is correct.*

Proof. This theorem follows directly from the specification of the scheme. □

Theorem 3 *Under the DBDH assumption, the DAA scheme specified in Section 4 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 DBDH 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 DBDH problem. Let $(g, g^{\mathsf{a}}, g^{\mathsf{b}}, g^{\mathsf{c}}, A = e(g, g)^{\mathsf{abc}}, B = e(g, g)^{\mathsf{d}}$, where $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathbb{Z}_q^*$ be the instance of the DBDH problem that we wish to answer which from A and B is equal to $e(g, g)^{\mathsf{abc}}$.

We now describe the construction of the simulator $\mathcal{S}$. $\mathcal{S}$ performs the following game with $\mathcal{A}$, as defined in Section 2.2.2. In the initial of the game, $\mathcal{S}$ runs `Setup` (or takes $\mathcal{A}$'s input) to get $\mathcal{I}$'s public key $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ (which is part of `params`) and secret key, namely `isk`, (x, y). Make all the values known to $\mathcal{A}$.

$\mathcal{S}$ creates algorithms to respond to queries made by $\mathcal{A}$ during its attack, including three random oracles denoted by H_1, H_2 and H_3, which refer to the hash-functions H used in $SPK\{(f) : F = g^f\}$, H_{G} used in computing B and H used in $SPK\{(r, f) : \mathsf{v}_s^r = \mathsf{v}_x \mathsf{v}_{xy}^f \wedge \mathsf{K} = \mathsf{B}^f\}(m)$, respectively. Note that the hash function H used to compute the value f does not have to be a random oracle, since it is an internal function.

To maintain consistency between queries made by $\mathcal{A}$, $\mathcal{S}$ keeps the following lists: L_i for $i = 1, 2, 3$ stores data for query/response pairs to random oracle H_i. L_{jc} stores data for query/response records for Join queries and Corrupted queries. Each item of L_{jc} is $\{ID, f, F, \texttt{cre}, c\}$, where $c = 1$ means that the corresponding signer is corrupted and $c = 0$ otherwise. L_s stores data for query/response records for Sign queries. Each item of L_s is $\{ID, m, \texttt{bsn}, \sigma, s\}$, where $s = 1$ means that $\texttt{bsn} = \perp$ and $s = 0$ means that $\texttt{bsn} \neq \perp$ 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: $H_1(m)$**.** If $(m, h_1) \in L_1$, return h_1. Else choose h_1 uniformly at random from $\mathbb{Z}_q^*$; add (m, h_1) to L_1 and return h_1.

Simulator: $H_2(m)$**.** If m has already been an entry of the H_2 query, i.e. the item (m, w, h_2) for an arbitrary w and h_2 exists in L_2, return h_2. Else choose v from $\mathbb{Z}_q^*$ uniformly at random; compute $h_2 := e(g^{\mathsf{b}}, g^{\mathsf{c}})^v$; add (m, v, h_2) to L_2 and return h_2.

Simulator: $H_3(m)$**.** If $(m, h_3) \in L_3$, return h_3. Else choose h_3 uniformly at random from $\mathbb{Z}_q^*$; add (m, h_3) to L_3 and return h_3.

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.

- If $i = \alpha$, choose u_α from $\mathbb{Z}_q^*$ uniformly at random; set $F_\alpha \leftarrow (g^{\mathsf{a}})^{u_\alpha}$; run $\mathsf{Join_t}$ with $\mathcal{A}$ to get $\texttt{cre}_\alpha$, and add $\{ID_\alpha, u_\alpha, F_\alpha, \texttt{cre}_\alpha, 0\}$ to L_{jc}. Note that since $\mathcal{S}$ does not know the value $f_\alpha = \mathsf{a}u_\alpha$, it is not able to execute as the prover in $SPK\{(f) : F_\alpha = g^f\}$. However $\mathcal{S}$ can forge the proof by controlling the random oracle of H_1 as follows: randomly choose s_f and $\mathfrak{c}$ and compute $\tilde{T} = g^{s_f} F^{-\mathfrak{c}}$. The only thing $\mathcal{S}$ has to take care of is checking the consistence of the L_1 entries. $\mathcal{S}$ verifies the validation of $\texttt{cre}_\alpha$ before accepting it.

- If $i = \beta$, choose u_β from $\mathbb{Z}_q^*$ uniformly at random; set $F_\beta \leftarrow (g^{\mathsf{a}})^{u_\beta}$; do the same thing as in the previous item to get $\texttt{cre}_\beta$.

- Else choose f uniformly at random from $\mathbb{Z}_q^*$; compute $F = g^f$, if $F = g^{\mathsf{a}}$ or g^{b} or g^{c}, abort outputting "**abortion 0**"; run $\mathsf{Join_t}$ with $\mathcal{A}$ to get $\texttt{cre}$; verify $\texttt{cre}$ before accept it and then add $(ID, f, F, \texttt{cre}, 0)$ 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

$(ID, f, F, \mathtt{cre}, 0)$ in L_{jc}, return f and update the item to $(ID, f, F, \mathtt{cre}, 1)$.

Simulator: Sign($ID, m, \mathtt{bsn}$**).** Let m' be the input message $\mathcal{A}$ wants to sign, $n_V \in \{0,1\}^{\ell_H}$ be a nonce chosen by $\mathcal{A}$ and $n_T \in \{0,1\}^{\ell_\phi}$ 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: $ID \neq ID_\alpha$ and $ID \neq ID_\beta$. Find the entry $(ID, f, F, \mathtt{cre}, 0/1)$ in L_{jc}, compute $\sigma \leftarrow \mathsf{Sign}$, add $(ID, m, \mathtt{bsn}, \sigma, 1/2)$ to L_s and respond with σ.

Case 2: $ID = ID_\alpha$. $\mathcal{S}$ 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_2 and H_3. $\mathcal{S}$ finds the entry $(ID_\alpha, u_\alpha, F_\alpha, \mathtt{cre}_\alpha = (a, b, c), 0)$ in L_{jc}, and forges σ by performing the following steps:

1. When $\mathtt{bsn} = \perp$, choose a random r; search whether r is an entry of L_2; if yes, go back to the beginning of this item. When $\mathtt{bsn} \neq \perp$, take the given $\mathtt{bsn}$, search whether $\mathtt{bsn}$ is an entry of L_2; if yes, retrieve the corresponding v and $h_2 = e(g^{\mathsf{b}}, g^{\mathsf{c}})^v$ values. With a new input of L_2, query H_2 to get v and h_2.
2. Set $\mathsf{B} := h_2$ and $\mathsf{K} := A^{u_\alpha v}$.
3. Choose random $r' \leftarrow \mathbb{Z}_q$ and compute $a' := a^{r'}$ and $b' := b^{r'}$.
4. Choose $c' \in \mathsf{G}$ at random.
5. Compute $\mathsf{v}_x := e(X, a')$, $\mathsf{v}_{xy} := e(X, b')$, $\mathsf{v}_s := e(g, c')$.
6. Choose $s_r, s_f \in \mathbb{Z}_q$ at random.
7. Choose $\mathfrak{c}$ at random; search whether $\mathfrak{c}$ is an entry of L_3; if yes, go back to the beginning of this item.
8. Compute $\tilde{\mathsf{T}}_1 := \mathsf{v}_s^{s_r} \mathsf{v}_{xy}^{-s_f} \mathsf{v}_x^{-\mathfrak{c}}$, and $\tilde{\mathsf{T}}_2 := \mathsf{B}^{s_f} \mathsf{K}^{-\mathfrak{c}}$.
9. Set $w = H(q \| g \| \mathsf{g} \| X \| Y \| a' \| b' \| c' \| \mathsf{v}_x \| \mathsf{v}_{xy} \| \mathsf{v}_s \| \mathsf{B} \| \mathsf{K} \| n_V) \| \tilde{\mathsf{T}}_1 \| \tilde{\mathsf{T}}_2 \| n_T \| m$; search whether $(w, \mathfrak{c})$ is an entry of L_3; if yes, go back to the beginning of the item of choosing s_r and s_f; otherwise, add $(w, \mathfrak{c})$ in L_3.
10. Output $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$.
11. Add $(ID_\alpha, m, \mathtt{bsn}, \sigma, 1/0)$ to L_s.

Case 3: $ID = ID_\beta$. Again, Si 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 $\mathsf{K} = B^{u_\beta v}$.

At the end of Phase 1, $\mathcal{A}$ outputs a message m, a basename $\mathtt{bsn}$, 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_0 and ID_1 by $\mathcal{A}$. If this is not the case we can define Join at these points as we wish i.e. as $F_\alpha = (g^{\mathsf{a}})^{u_\alpha}$ and $F_\beta = (g^{\mathsf{a}})^{u_\beta}$ where $u_\alpha, u_\beta \in \mathbb{Z}_q^*$ is chosen uniformly at random. Neither ID_0 nor ID_1 should have been asked for the Corrupt query and the Sign query with the same $\mathtt{bsn} \neq \perp$ by following the definition of the game defined in Section 2.2.

$\mathcal{S}$ chooses a bit b at random, and generates the challenge by querying Sign$(ID_\alpha, m, \texttt{bsn})$ if $b = 0$ or Sign$(ID_\beta, m, \texttt{bsn})$ otherwise in the same way as Case 2 of the Sign query simulation. $\mathcal{S}$ 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 $\texttt{bsn} \neq \perp$. At the end of Phase 2, $\mathcal{A}$ outputs b', $\mathcal{S}$ considers the following 4 cases:

- Case 1. If $b = b' = 0$, $\mathcal{S}$ marks "true-A".
- Case 2. If $b = b' = 1$, $\mathcal{S}$ marks "true-B".
- Case 3. If $b = 0, b' = 1$, $\mathcal{S}$ marks "failure-A".
- Case 4. If $b = 1, b' = 0$, $\mathcal{S}$ marks "failure-B".

$\mathcal{S}$ runs the above game with $\mathcal{A}$ k times. At the end of the k games, the number of $b = 0$ and the number of $b = 1$ should be identical, based on the random selection of b. $\mathcal{S}$ sets the numbers of "true-A" and "true-B" as k_A and k_B respectively. If $k_A = k_B$, $\mathcal{S}$ aborts outputting "**abortion 2**". If $k_A > k_B$, $\mathcal{S}$ answers that $A = e(g, g)^{\mathsf{abc}}$ holds; if $k_A < k_B$, $\mathcal{S}$ answers that $B = e(g, g)^{\mathsf{abc}}$ holds.

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_1, H_2 and H_3 are indistinguishable from real random oracles.

If the event **abortion 0** happens, $\mathcal{S}$ gets the value a or b or c, $\mathcal{S}$ can compute $e(g, g)^{\mathsf{abc}}$ and thus to solve the DBDH problem (because the DBDH problem is weaker than the BDH problem). Since $\mathcal{S}$ chooses its value uniformly at random from $\mathbb{Z}_q^*$, the chance of this event happens is negligible.

The event **abortion 1** happens 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 $1/q_j(q_j - 1)$.

The event **abortion 2** happens if $k_A = k_B$. We argue that this case is confliction to the assumption that $\mathcal{A}$ succeeds with a non-negligible probability to break user-controlled-anonymity. In order to break user-controlled-anonymity, $\mathcal{A}$ must find the link between $(F, \texttt{cre})$ and σ w.r.t. the identity of the corresponding signer. In the DAA scheme, $F = g^f$, $\texttt{cre} = (a, b, c)$ and $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$, where $\mathsf{K} = \mathsf{B}^f$, $a' = a^{r'}$, $b' = b^{r'}$ and $c' = c^{r'r^{-1}}$. Since the values $r', r \in \mathbb{Z}_q^*$ are chosen uniformly at random, the triplet (a', b', c') must be indistinguishable from the triplet (a, b, c) from the view of any entity with the polynomial computational capability. The values $\mathfrak{c}$, s_r and s_f are uniformly and randomly distributed in $\mathbb{Z}_q^*$. Therefore, to win the game with the probability larger than $1/2$, $\mathcal{A}$ must be able to find whether $\log_g F = \log_{\mathsf{B}} \mathsf{K}$ holds. The fact that $b = 0$ and "true-A" is marked means that $\mathcal{A}$ has found out that $\log_g g^{\mathsf{a}} = \log_{e(g,g)^{\mathsf{bc}}} A$ holds. The fact that $b = 1$ and "true-B" is marked means that $\mathcal{A}$ has found out that $\log_g g^{\mathsf{a}} = \log_{e(g,g)^{\mathsf{bc}}} B$ holds. But in the instance of the DBDH problem, only A or B not both is the correct figure.

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.

Observe that if the random oracle $H_{\mathsf{G}}(0\|\texttt{bsn}_I)$ is used in Join as the same as in the original DAA scheme [8], the notion of user-controlled-anonymity in the proposed DAA scheme can be proved under the DDH assumption instead of the DBDH assumption as follows: given the instance of the DDH problem $(\mathsf{g}, \mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}}, A = \mathsf{g}^{\mathsf{ab}}, B = \mathsf{g}^{\mathsf{c}})$, $\mathcal{S}$ can use this oracle to set $N_I = H_{\mathsf{G}}(0\|\texttt{bsn}_I)^f = \mathsf{g}^{\mathsf{a}}$, $\mathsf{B} = \mathsf{g}^{\mathsf{b}}$, $\mathsf{K}_0 = A$ and $\mathsf{K}_1 = B$, and then can take the result of guessing $\log_{\mathsf{g}} N_I = \log_{\mathsf{g}^{\mathsf{b}}} \mathsf{K}_b$ for $b = \{0, 1\}$ from $\mathcal{A}$ to find out which one from A and B is g^{ab}. □

Theorem 4 *Under the LRSW assumption, the DAA scheme specified in Section 4 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 LRSW 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 LRSW problem. Let $X, Y \in G, X = g^x, Y = g^y$ and $(a, a^y, a^{x+xym}) \leftarrow \mathcal{O}(m, x, y)$ be the instance of the LRSW problem that we wish to provide $(\tilde{m}, \tilde{a}, \tilde{b}, \tilde{c})$ such that $m \neq 0$, $\tilde{b} = \tilde{a}^y$ and $\tilde{c} = \tilde{a}^{x+xy\tilde{m}}$ where $\tilde{m}$ has not be queried to the oracle $\mathcal{O}$ before.

We now describe the construction of the simulator $\mathcal{S}$. $\mathcal{S}$ performs the following game with $\mathcal{A}$, as defined in Section 2.2.3. In the initial of the game, $\mathcal{S}$ sets $\mathcal{I}$'s public key as $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ (which is part of `params`) and secret key, namely `isk`, as (x, y). $\mathcal{S}$ gives `params` to $\mathcal{A}$. 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_1, H_2 and H_3 in the same way as in the proof of Theorem 3. To maintain consistency between queries made by $\mathcal{A}$, $\mathcal{S}$ keeps the following lists: L_i for $i = 1, 2, 3$ stores data for query/response pairs to random oracle H_i. L_{jc} stores data for query/response records for Join queries and Corrupted queries. Each item of L_{jc} is $\{ID, f, F, \texttt{cre}, c\}$, 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 `ROGUE` list. L_s stores data for query/response records for Sign queries. Each item of L_s is $\{ID, m, \texttt{bsn}, \sigma, s\}$, where $s = 1$ means that $\texttt{bsn} = \perp$ under the Sign query and $s = 0$ means that $\texttt{bsn} \neq \perp$ 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 the time values are polynomial.

Simulator: $H_1(m)$**.** The same as in the proof of Theorem 3.

Simulator: $H_2(m)$**.** If m has already been an entry of the H_2 query, return h_2. Else choose h_2 from $\mathbb{Z}_q^*$ uniformly at random and return h_2.

Simulator: $H_3(m)$**.** The same as in the proof of Theorem 3.

Simulator: Join(ID**).** We assume $\mathcal{A}$ does not make repeat queries. Given a new ID from $\mathcal{A}$ (Case 1), $\mathcal{S}$ chooses $f \in \mathbb{Z}_q^*$ uniformly at random; or $\mathcal{S}$ receives a new pair of ID and f from $\mathcal{A}$ (Case 2). Then in both cases, $\mathcal{S}$ asks $\mathcal{O}$ to provide $\texttt{cre} = (a, b = a^y, c = a^{x+xyf})$, adds $\{ID, f, F, \texttt{cre}, 0\}$ to L_{jc} in Case 1 and $\{ID, f, F, \texttt{cre}, 1\}$ to L_{jc} in Case 2, and returns `cre` to $\mathcal{A}$.

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 $(ID, f, F, \texttt{cre}, 0)$ in L_{jc}, return f and update the item to $(ID, f, F, \texttt{cre}, 1)$.

Simulator: Sign($ID, m, \texttt{bsn}$**).** Let m' be the input message $\mathcal{A}$ wants to sign, $n_V \in \{0,1\}^{\ell_H}$ be a nonce chosen by $\mathcal{A}$ and $n_T \in \{0,1\}^{\ell_\phi}$ 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 $(ID, f, F, \texttt{cre}, 0/1)$ in L_{jc}, compute $\sigma \leftarrow \mathsf{Sign}$, add $(ID, m, \mathtt{bsn}, \sigma, 1/0)$ to L_s and respond with σ.

Simulator: Semi-sign($ID, m, \mathsf{B}, \mathsf{v}_{xy}, \tilde{T}_{1t}, \mathsf{c}_H$). 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 $(ID, f, F, \mathtt{cre}, 0/1)$ in L_{jc}, compute $(\mathsf{K}, \mathsf{c}, s_f)$ by following the $\mathcal{M}$'s action in Sign, add $(ID, m, \mathtt{bsn}, \sigma = (\mathsf{B}, \mathsf{K}, *, *, *, \mathsf{c}, *, s_f), *)$ to L_s and respond with $(\mathsf{K}, \mathsf{c}, s_f)$.

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

- Case 1. If $\mathsf{Verify}(\sigma) = 1$ and $(ID, m, \mathtt{bsn}, \sigma, 1/0)$ (or $(ID, m, \mathtt{bsn}, \sigma = (\mathsf{B}, \mathsf{K}, *, *, *, \mathsf{c}, *, s_f), *)$) is not in L_s, $\mathcal{S}$ rewinds $\mathcal{A}$ to extract the knowledge of r and f, satisfying $\mathsf{v}_s^r = \mathsf{v}_x \mathsf{v}_{xy}^f$, from $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$. The triplet (a', b', c'^r) is a Camenisch-Lysyanskaya signature on the message f. Since the value $f \notin \mathtt{ROGUE}$ (implied in $\mathsf{Verify}(\sigma) = 1$), the signature is not a result of the $\mathcal{O}$ query. Following Theorem 1, (a', b', c'^r) is a right solution of the LRSW problem. $\mathcal{S}$ solves the problem.
- Case 2. Suppose $\mathtt{bsn} \neq \perp$. If no any entry $(ID, m', \mathtt{bsn}, \sigma', 1/0)$ 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 $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$ and $\sigma' = (\mathsf{B}, \mathsf{K}', a'', b'', c'', \mathfrak{c}', s'_r, s'_f)$. Both σ and σ' have B since they have the same $\mathtt{bsn}$ and $\mathcal{S}$ has maintained the consistence of the random oracle H_2 outputs. The only thing to make $\mathsf{K} \neq \mathsf{K}'$ happen is that $\mathcal{A}$ has managed to create a different $\mathtt{tsk}$ for ID. Then $\mathcal{S}$ can use the same trick as in Case 1 to extract a right solution of the LRSW problem from $\mathcal{A}$.

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

6 Consideration on Implementing the DAA Scheme

In this section, we show how to implement the proposed DAA scheme. We first recall a well-known construction of an admissible bilinear map from the Tate pairing, as our recommended example of pairings, describe how to choose the corresponding security parameters, and then analyze the performance of our DAA scheme when uses this bilinear map. Finally we discuss various constructions of the point B in the DAA scheme.

6.1 An admissible bilinear map from the Tate pairing

We now recall the description of an admissible bilinear map [3, 22, 25] from the Tate pairing. Let p be a prime satisfying $p = 3 \bmod 4$ and let q be some prime factor of $p+1$. Let E be the elliptic curve defined by the equation $y^2 = x^3 - 3x$ over $\mathbb{F}_p$. $E(\mathbb{F}_p)$ is supersingular and contains $p+1$ points and $E(\mathbb{F}_{p^2})$ contains $(p+1)^2$ points. Let $g \in E(\mathbb{F}_p)$ to a point of order q and let G be the subgroup of points generated by g. Let G be the subgroup of $\mathbb{F}_{p^2}^*$ of order q.

Let $\phi(x, y) = (-x, iy)$ be an automorphism of the group of points on the curve $E(\mathbb{F}_p)$, where $i^2 = 1$. Then ϕ maps points of $E(\mathbb{F}_p)$ to points of $E(\mathbb{F}_{p^2}) \backslash E(\mathbb{F}_p)$. Let f be the Tate pairing, then we can define $e : G \times G \rightarrow \mathsf{G}$ as $e(P, Q) = f(P, \phi(Q))$, where e is an admissible bilinear map.

6.2 Choices of security parameters

To choose right sizes of p and q, we must ensure that the discrete log problem in G is hard enough. As the discrete log problem in G is efficiently reducible to discrete log in G [27], we need to choose p large enough such that the discrete log in $\mathbb{F}^*_{p^2}$ is hard to compute. For our DAA scheme, we choose p as a 512-bit prime and q as a 160-bit prime as follows [25]: (I) Choose a 160-bit prime q. Careful choices of q would speed up the Tate pairing operation substantially, e.g., choose q with low Hamming weight. (II) Randomly generate a 352-bit r where $4 \mid r$. (III) Compute $p := rq - 1$ and check whether p is a prime. Note that $p = 3 \bmod 4$. If p is not prime, repeat the previous step.

6.3 Efficiency of our DAA scheme

Let $\ell_H = 256$, $\ell_p = 512$, $\ell_q = 160$ and $\ell_\phi = 80$. Given a concrete DAA scheme described above using Tate pairing, we summarize the performance of our scheme as follows:

- Since p is 512-bit, we can use 513 bits to present a point in $E(\mathbb{F}_p)$. The size of the DAA public key is 2211 bits or 277 bytes. Each membership private key is 1699 bits or 213 bytes. Each signature is 4163 bits or 521 bytes.
- To compute a signature, the TPM needs to perform 5 exponentiations and the host needs to perform 5 exponentiations and 3 pairings. To verifier a signature, the verifies needs to 7 exponentiations and 5 pairings.

Note that the host can pre-compute $A = e(X, a)$, $B = e(X, b)$ and $C = e(g, c)$, and store the triple (A, B, C). In each signing process, the host can compute $\mathsf{v}_x := A^{r'}$, $\mathsf{v}_{xy} := B^{r'}$, and $\mathsf{v}_s := C^{r'r^{-1}}$ to avoid expensive pairing operations.

6.4 Constructing the Point B

How to construct $\mathsf{B} \xleftarrow{R} \mathsf{G}$ and $\mathsf{B} = H_{\mathsf{G}}(m)$ given the input m is a sensitive part of the scheme implementation. To implement $\mathsf{B} \xleftarrow{R} \mathsf{G}$, we suggest creating a random generator of G, which is a q-th root of unity in $\mathbb{F}_{p^2}$. This can be done by choosing a random $x \in \mathbb{F}_{p^2}$, then computing $\mathsf{B} := x^{(p^2-1)/q}$. We suggest the following various ways to construct $\mathsf{B} = H_{\mathsf{G}}(m)$.

1. Use a collision resistant hash function $H(m)$, which maps the value m to an element in F_{p^2}, then compute $\mathsf{B} = H(m)^{(p^2-1)/q}$.
2. Use a MapToPoint function H, as described in [7], to map the value m to an element of G, that can guarantee none knows the discrete log relation between g and $H(m)$, and then compute $\mathsf{B} := e(H(m), H(m))$.
3. As in the original DAA scheme [8], make a cyclic group different from either G or G, in which the discrete logarithm problem is hard, and then compute B in this group instead of G.

7 Conclusion

In this paper, we have introduced the formal definitions of two security notions for DAA, namely user-controlled-anonymity and user-controlled-traceability. This is a simplified version of the formal security model provided in [8]. We have also proposed a new DAA scheme from pairings, which requires a much shorter key length compared with the original integer factorization based DAA scheme. We have proved the security of the new DAA scheme under the proposed security notions.

Acknowledgements

We would like to thank Frederik Armknecht, Hans Loehr, Paul Morrissey, Kenny Paterson, Mark Ryan, Ahmad-Reza Sadeghi, Nigel Smart, Ben Smyth and Robert Squibbs for helpful discussions and comments on this work.

References

[1] M. backes, M. maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestaion protocol. Cryptology ePrint Archive, Report 2007/289, 2007. `http://eprint.iacr.org/`.

[2] S. Balfe, A. D. Lakhani, and K. G. Paterson. Securing peer-to-peer networks using trusted computing. In C. Mitchell, editor, *Chapter 10 of Trusted Computing*, pages 271–298. IEE, London, 2005.

[3] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology — CRYPTO '02*, volume 2442 of *LNCS*, pages 354–368. Springer, 2002.

[4] 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 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, 2003.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.

[6] D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology — EUROCRYPT '04*, volume 3027 of *LNCS*, pages 223–238. Springer, 2004.

[7] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology — CRYPTO '01*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.

[8] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[9] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation in context. In C. Mitchell, editor, *Chapter 5 of Trusted Computing*, pages 143–174. IEE, London, 2005.

[10] E. F. Brickell, D. Chaum, I. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *LNCS*, pages 156–166. Springer, 1987.

[11] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In C. Blundo and S. Cimato, editors, *Proceedings of Forth International Conference on Security in Communication Networks, SCN 2004*, volume 3352 of *LNCS*, pages 122–135. Springer, 2005.

[12] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Proceedings of the 3rd Conference on Security in Communication Networks*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[13] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology — CRYPTO '04*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.

[14] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *In Advances in Cryptology — CRYPTO '99*, volume 1666 of *LNCS*, pages 413–430. Springer, 1999.

[15] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO '97*, volume 1296 of *LNCS*, pages 410–424. Springer, 1997.

[16] D. Chaum. Zero-knowledge undeniable signatures. In *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *LNCS*, pages 458–464. Springer, 1990.

[17] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Advances in Cryptology — EUROCRYPT '87*, volume 304 of *LNCS*, pages 127–141. Springer, 1987.

[18] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology — CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1992.

[19] I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology — ASIACRYPT '02*, volume 2501 of *LNCS*, pages 125–142. Springer, Dec. 2002.

[20] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[21] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology — CRYPTO '97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

[22] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In *Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer.

[23] H. Ge and S. R. Tate. A direct anonymous attestation scheme for embedded devices. In *Proceedings of Public Key Cryptography - PKC 2007*, volume 4450 of *LNCS*. Springer, 2007.

[24] A. Leung and C. J. Mitchell. Ninja: Non identity based, privacy preserving authentication for ubiquitous environments. In *Proceedings of 9th International Conference on Ubiquitous Computing*, volume 4717 of LNCS, pages 73–90. Springer-Verlag, 2007.

[25] B. Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University, Stanford, California, 2007.

[26] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Proceedings of the 6th Workshop on Selected Areas in Cryptography*, volume 1758 of *LNCS*, pages 184–199. Springer, 1999.

[27] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the 23rd annual ACM Symposium on Theory of Computing (STOC)*, pages 80–89. ACM Press, 1991.

[28] A. Pashalidis and C. J. Mitchell. Single sign-on using TCG-conformant platforms. In C. Mitchell, editor, *Chapter 6 of Trusted Computing*, pages 175–193. IEE, London, 2005.

[29] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

[30] C. Rudolph. Covert identity information in direct anonymous attestation (DAA). In *Proceedings of the 22nd IFIP TC-11 International Information Security Conference (SEC2007)*, 2007.

[31] C. P. Schnorr. Efficient identification and signatures for smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[32] B. Smyth, L. Chen, and M. Ryan. Direct anonymous attestation (DAA): ensuring privacy with corrupt administrators. In F. Stajano, editor, *Proceedings of Fourth European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2007)*, volume 4572 of LNCS, pages 218–231. Springer-Verlag, 2007.

[33] Trusted Computing Group. TCG TPM specification 1.2, 2003. Available at `http://www.trustedcomputinggroup.org`.

[34] Trusted Computing Group website. `http://www.trustedcomputinggroup.org`.