Pairings in Trusted Computing

Chen, Liqun; Morrissey, Paul; Smart, Nigel P
HP Laboratories
HPL-2008-73

Keyword(s):
Trusted computing, Direct Anonymous Attestation (DAA), pairing

Abstract:
Pairings have now been used for constructive applications in cryptography for around eight years. In that time the range of applications has grown from a relatively narrow one of identity based encryption and signatures, through to more advanced protocols. In addition implementors have realised that pairing protocols once presented can often be greatly simplified or expanded using the mathematical structures of different types of pairings. In this paper we consider another advanced application of pairings, namely to the Direct Anonymous Attestation (DAA) schemes as found in the Trusted Computing Group standards. We show that a recent DAA proposal can be further optimized by transferring the underlying pairing groups from the symmetric to the asymmetric settings. This provides a more efficient and scalable solution than the existing RSA and pairing based DAA schemes.

External Posting Date: June 21, 2008 [Fulltext] Approved for External Publication

Internal Posting Date: June 21, 2008 [Fulltext]

To be published in The Second International Conference on Pairing Cryptography Proceedings, Sept 1-3, 2008

Pairings in Trusted Computing*

L. Chen[1], P. Morrissey[2] and N.P. Smart[2]

[1] Hewlett-Packard Laboratories,
Filton Road,
Stoke Gifford,
Bristol, BS34 8QZ,
United Kingdom.
liqun.chen@hp.com
[2] Computer Science Department,
Woodland Road,
University of Bristol,
Bristol, BS8 1UB,
United Kingdom.
{paulm, nigel}@cs.bris.ac.uk

Abstract. Pairings have now been used for constructive applications in cryptography for around eight years. In that time the range of applications has grown from a relatively narrow one of identity based encryption and signatures, through to more advanced protocols. In addition implementors have realised that pairing protocols once presented can often be greatly simplified or expanded using the mathematical structures of different types of pairings. In this paper we consider another advanced application of pairings, namely to the Direct Anonymous Attestation (DAA) schemes as found in the Trusted Computing Group standards. We show that a recent DAA proposal can be further optimized by transferring the underlying pairing groups from the symmetric to the asymmetric settings. This provides a more efficient and scalable solution than the existing RSA and pairing based DAA schemes.

1 Introduction

The growth of pairing based cryptography and the growth of elliptic curve cryptography from an implementation perspective closely follow the same path. Originally elliptic curve systems were defined over supersingular elliptic curves. This was mainly due to the difficulty in constructing suitable curves with a known group order of the required size, and also due to perceived performance advantages which accrue from using supersingular curves. With the discovery of the MOV attack [20], implementors moved over to using non-supersingular (or ordinary) elliptic curves. This move was supported by the research conducted into the Schoof algorithm and its variants, [2]. The Schoof algorithm enabled ordinary

* The second and third author would like to thank EPSRC for partially supporting the work in this paper.

elliptic curves to be constructed with a known number of group elements, thus enabling standard elliptic curve cryptography to be performed using ordinary elliptic curves.

Pairing based cryptography also started by using supersingular elliptic curves, via the use of symmetric pairings. Again this was mainly because such curves enabled one to compute the number of points very efficiently, and because it appeared that symmetric pairings could be implemented much more efficiently than standard pairings. However, a major drawback of symmetric pairings is that their security properties scale badly. This poor scaling is due to the embedding degree being bounded by six. Thus with the wider acceptance of AES style security levels it has been necessary for pairing protocols to also move to the setting of ordinary elliptic curves, where asymmetric pairings are required. This security concern, which has prompted the move to asymmetric pairings, has been supported by a large body of research into optimizing pairings in the ordinary elliptic curve setting, and in generating the required parameters. Probably, at the time of writing the best choice for parameters is to choose a Barreto-Naehrig curve [1], implement the Ate-pairing [18], and use sextic twists to reduce the complexity of the group operations.

However, whilst protocols in the standard elliptic curve cryptography setting move seamlessly from the supersingular case to the ordinary case, this is not true in the pairing based cryptography setting. For example, issues arise with respect to hashing onto various groups, or from mappings between the two groups in the domain, see [15]. Thus in pairing based cryptography various initial protocol suggestions often needed to be revisited as asymmetric pairings became more accepted as the default implementation choice. For example the original Boneh-Franklin encryption scheme [6] was originally presented for symmetric pairings and in this setting is highly efficient. However, at high security levels it is less attractive than some of the more modern approaches such as the Boneh-Boyen scheme [3] (which is preferred by those who worry about exact security) and the SK-KEM scheme [13] (which has a better performance than the Boneh-Boyen scheme, but worse exact security). Another example of the need to fully evaluate pairings in the asymmetric setting can be found in ID-based key agreement, for which [14] provides a good summary of the issues involved.

Over the years various more advanced protocols have been proposed which use pairings; for example encryption with keyword search [5], group signatures [4], traitor tracing [19]. In this paper we consider another advanced application of pairings, namely to Direct Anonymous Attestation (DAA) schemes as found in the Trusted Computing Group standards [21]. We show that a recent DAA proposal [8] can be further optimized, and hence we provide a more efficient and scalable solution than the existing RSA and pairing based DAA schemes.

The original DAA scheme [7] was based on a signature scheme of Camenisch and Lysyanskaya [10] whose security was based on the strong-RSA assumption and the decisional Diffie–Hellman assumption in a finite field. In [16] another DAA scheme was presented, based on the Camenisch and Michels signature

scheme [12], this again results in a DAA scheme which is secure under the strong-RSA and the decisional Diffie–Hellman assumption in a finite field.

Recently, Brickell et. al. [8] have presented a DAA scheme based on symmetric pairings. This DAA protocol is based on another signature scheme of Camenisch and Lysyanskaya [11] which makes use of symmetric pairing groups. This results in a scheme which is secure under the DBDH assumption and the LRSW assumption. This latter assumption is a non-standard assumption which underlies the Camenisch and Lysyanskaya signature scheme. The LRSW assumption was introduced in [17], where it was shown that it holds in the generic group model.

As a starting point we take the pairing based DAA scheme of Brickell et. al. and provide some efficiency improvements. In addition we present the scheme in the asymmetric setting, which requires, as is usual in this situation, some minor modifications to the original scheme. We show that the new scheme is particularly suited to the environment in which the DAA scheme is meant to run. This is because our new scheme places a smaller computational requirement on the TPM, which is a small hardware device which sits on the motherboard of the trusted platform. In fact the TPM has only to perform a single basic elliptic curve point multiplication in the signing protocol, and in all parts of the protocol the TPM requires no operation to be performed in large finite fields nor any pairing calculations. In addition we reduce the number of pairings computed by the Host during a signing operation from three to one.

In the full version of this paper we shall show that our optimized asymmetric pairing based protocol is secure in a security model based on real/ideal world simulations. This is closer to the original security model of [7], than the security model used in [8]. Thus our protocol is not only more efficient than that in [8], but it also enjoys enhanced security properties.

2 Introduction to Direct Anonymous Attestation

In order to give an intuitive explanation of what direct anonymous attestation (DAA) is, its importance and its impact, we use the following scenario. Consider a user, Alice, who owns a laptop computer. Alice uses this computer for online shopping with a given retailer Charlie and to remotely log on to network server Bob in order to work from home as part of her day job. Both Bob and Charlie want some assurance that Alice is using a laptop which contains some combination of hardware and software from some specific set. This is to protect themselves from malicious users who may try to compromise their systems. In other words they want some assurance that Alice's platform can be trusted. On the other hand Alice does not want either Bob or Charlie to know exactly which laptop she owns, what hardware it contains, or what software it runs: just that it can in fact be trusted. Furthermore, Alice wants Bob to be able to link transactions made with her laptop to each other (without giving Bob any more information than that the transactions were made using the same platform), but does not want any other transactions to be linked. Informally, a DAA scheme is a mechanism for achieving each of these seemingly contradictory goals.

We assume that each trusted platform has a certain module, known as a trusted platform module (TPM), embedded into it at the time of manufacture. Each TPM will have a unique endorsement key pair (EK) which is also chosen at the time of manufacture and which is hidden inside the TPM. Usually the TPM is a small chip embedded onto a computer's motherboard and as a result the TPM has only limited computational and storage resources. As such the TPM is a potential bottleneck within a DAA scheme. To make a DAA scheme as efficient as possible the main goal is to minimize the amount of computation that a given TPM will have to perform. We refer to the platform into which the TPM is embedded as the Host and the combination of TPM and Host as a *user*.

In order to convince a verifier that a platform contains such a TPM, and can hence be trusted, the user has to first obtain a credential from some credential issuer. It does this by having its TPM compute a commitment to a secret internal value f that is unique for each issuer/TPM pair. This commitment is then used as evidence to the issuer that the user does in fact have a valid TPM embedded within it. Note that a given user can obtain many credentials from a given issuer.

In order to convince a given verifier that a user owns such a credential the user computes a "signature of knowledge" of such a credential and the associated value f corresponding to this credential. This signature of knowledge is then sent to the verifier. Then, since the credential was issued by a specific issuer, the verifier is convinced that a given user contains a valid TPM but does not know which TPM this is. If a user wants a verifier to be able to link transactions then that user simply computes the signature of knowledge in a certain specified way: by using a given verifier basename.

One last consideration is what happens if a given TPM is compromised and its secret internal values published? In this case we use rogue tagging. Each issuer and verifier maintains their own list of rogue values. When a given value of f is published they then decide whether to add it to their list or not. Then we require, when a user computes a signature of knowledge of a given credential this includes some information that allows for the signature to be recognised as produced by a compromised TPM secret value

Since the verifier is only given a signature of knowledge of a credential, and not the credential itself, if the issuer and the verifier that computed the credential collude, then they should not be able to identify transactions made by a specific TPM. Yet a given verifier will be assured that any transaction that does take place was made by a platform that contains such a TPM and that this TPM has not been compromised.

2.1 The DAA Players

We refer to each of the entities in a DAA scheme as *players*. We first describe the types of players we consider in our model. This set of players is the same set as in [9] and is intended to represent a DAA scheme in which a given TPM wishes to remotely and anonymously authenticate itself to a given verifier. Intuitively, the set of players will consist of a set of users, each comprising a Host and a

TPM, a set of issuers, and a set of verifiers to which users want to authenticate their TPM.

We now give a formal description of each of the DAA players. A general DAA scheme has a set of players that consists of the following.

- A set of users $\mathcal{U}$ where each $U_i \in \mathcal{U}$ consists of
  - A TPM m_i from some set of TPMs $\mathcal{M}$ with an endorsement key ek_i and seed $\mathsf{DaaSeed}_i$;
  - A Host h_i from some set of hosts $\mathcal{H}$ which will have a counter value cnt_i, a set of commitments $\{\mathsf{comm}\}_i$ and a set of credentials $\{\mathsf{cre}\}_i$.
- A set of issuers $\mathcal{I}$ where each $I_k \in \mathcal{I}$ has a public and private key pair $(\mathsf{ipk}_k, \mathsf{isk}_k)$ and long term value K_k (for example a long term public key of the issuer). Each $I_k \in \mathcal{I}$ also maintains a list of rogue TPM internal values, we denote this list by $\mathsf{RogueList}(I_k)$.
- A set of verifiers $\mathcal{V}$. Each verifier $V_j \in \mathcal{V}$ maintains a set of base names $\{\mathsf{bsn}\}_j$ and a list of rogue TPM internal values $\mathsf{RogueList}(V_j)$. Each V_j may optionally maintain a list of message and signature pairs received (this can be used to trade memory for computation in linking).

We assume that initially the sets $\{\mathsf{comm}\}_i$, $\{\mathsf{cre}\}_i$ are empty for all $U_i \in \mathcal{U}$. In addition we assume that the list $\mathsf{RogueList}(I_k)$ are empty for all $I_k \in \mathcal{I}$, and that the list $\mathsf{RogueList}(V_j)$ and the sets $\{\mathsf{bsn}\}_j$ are empty for all $V_j \in \mathcal{V}$.

It is worth describing the various player parameters and how they relate to each other. Generally, at the time of manufacture, each TPM will have a single endorsement key ek_i embedded into the TPM chip. In addition, each TPM generates a TPM-specific secret $\mathsf{DaaSeed}_i$ and stores it in nonvolatile memory, this value will never be disclosed or changed by the TPM. We do not consider choosing and assigning the values ek_i and $\mathsf{DaaSeed}_i$ in the setup algorithm, since the setup algorithm is run only by an issuer. The $\mathsf{DaaSeed}_i$ is generally a 20–byte constant that, together with a given issuer value K_k, allows for the generation and regeneration of a given value of an internal secret key f. Each TPM can have multiple possible values for f (at least one per issuer and possible more if a given issuer has more than one value of K_k). We refer to the set of possible values of f for a given user i as $\{f\}_i$ Since the TPM has limited storage requirements it does not store the current value for f, it regenerates it as required from $\mathsf{DaaSeed}_i$. For each value of f the TPM will be able to compute a single commitment on f. The value cnt_i that a given Host maintains can be thought of as an index for a particular f and commitment pair.

For each commitment, as we will see later, a given issuer could issue multiple credentials. We assume the Host only stores one credential for a given f/commitment pair, and hence the value of cnt_i will also refer to the *current* value of the corresponding credential.

The set $\{\mathsf{bsn}\}_j$ is used to achieve user controlled linkability of signatures.

2.2 Formal Definition of a DAA Scheme

Informally a DAA scheme consists of a system setup algorithm, a protocol for users to obtain credentials, a signing protocol, algorithms for verifying and link-

ing signatures and an algorithm for tagging rogue TPM values. Our definition is similar to that given in [8] but with some modifications. Specifically, we give a single protocol for the joining functionality as opposed to multiple protocols, and our signature functionality is given as a protocol as opposed to an algorithm. Also we have an additional rogue tagging algorithm.

Definition 1 (Daa Scheme). Formally, we define a Daa scheme to be a tuple of protocols and algorithms $\mathsf{Daa} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link}, \mathsf{RogueTag})$ where:

- $\mathsf{Setup}(1^t)$ is a p.p.t. system setup algorithm. On input 1^t, where t is a security parameter, this outputs a set of system parameters par which contains all of the issuer public keys ipk_k and the various parameter spaces. This algorithm also serves to setup and securely distribute each of the issuer secret keys isk_k.
- $\mathsf{Join}(U_i, I_k)$ is a 3 party protocol run between a TPM, a Host and an issuer. In a correct initial run of the protocol with honest players the Host should obtain an additional valid commitment and an additional valid credential. In correct subsequent runs one valid credential should be replaced with another.
- $\mathsf{Sign}(U_i, \mathsf{msg})$ is a 2 party protocol run between a TPM and a Host used to generate a signature of knowledge on some message msg. In a correct run of the protocol with honest players the signature of knowledge will be constructed according to some basename for some specified verifier that may or may not allow the signature to be linked to other signatures with this same verifier.
- $\mathsf{Verify}(\sigma, \mathsf{msg})$ is a deterministic polynomial time (d.p.t.) verification algorithm that allows a given verifier to verify a signature of knowledge σ of a credential on a message msg intended for a given verifier with a specific basename. The verification process will involve checking the signature against the list $\mathsf{RogueList}(V_j)$. This algorithm returns either *accept* or *reject*.
- $\mathsf{Link}(\sigma_0, \sigma_1)$ is a d.p.t. linking algorithm that returns either *linked*, *unlinked* or $\perp$. The algorithm should return $\perp$ if either signature was produced with a rogue key, return *linked* if both are valid signatures and the user who produced them wanted these to be linkable to each other, and return *unlinked* otherwise.
- $\mathsf{RogueTag}(f, \sigma)$ is a d.p.t. rogue tagging algorithm that returns true if σ is a valid signature produced using the TPM secret value f and returns false otherwise.

For correctness we require that if

- a user $U_i \in \mathcal{U}$ engages in a run of Join with I_k, resulting in U_i obtaining a commitment comm on a TPM secret value f and a credential cre corresponding to f,
- the user U_i then creates two signatures σ_b on two messages msg_b for $b \in \{0, 1\}$ intended for verifier $V_j \in \mathcal{V}$ with basename bsn (which could be $\perp$),
- and the secret TPM value used to compute these f is not in $\mathsf{RogueList}$.

Then

$$\mathsf{Verify}(\sigma_0, \mathsf{msg}_0) = \mathsf{Verify}(\sigma_1, \mathsf{msg}_1) = \mathit{accept}$$

and if $\mathsf{bsn} \neq \perp$ then $\mathsf{Link}(\sigma_0, \sigma_1) = \mathit{linked}$.

3 The Camensich-Lysyanskaya Signature Scheme

Before proceeding it is worth pausing to present the pairing based Camensich-Lysyanskaya signature scheme which is at the heart of not only our DAA scheme, but also the scheme of [8]. We let $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ denote a pairing between three groups of prime order q. We let the generator of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) be denoted by P_1 (resp. P_2).

- **KeyGeneration:** The private key is a pair $(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q$, the public key is given by the pair $(X, Y) \in \mathbb{G}_2 \times \mathbb{G}_2$ where $X = xP_2$ and $Y = yP_2$.
- **Signing:** On input of a message $m \in \mathbb{Z}_q$ the signer generates $A \in \mathbb{G}_1$ at random and outputs the signature $(A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1$, where $B = yA$ and $C = [x + mxy]A$.
- **Verification:** To verify a signature on a message the verifier checks whether $\hat{t}(A, Y) = \hat{t}(B, P_2)$ and $\hat{t}(A, X) \cdot \hat{t}(mB, X) = \hat{t}(C, P_2)$.

The original signature scheme is given in the symmetric pairing setting (i.e. where $\mathbb{G}_1 = \mathbb{G}_2$), we have chosen the above asymmetric version to reduce the size of the signatures and to have the fastest signing algorithm possible. The key property of this signature scheme is that signatures are re-randomizable without knowledge of the secret key: given (A, B, C) one can re-randomize it by computing (rA, rB, rC) for a random element $r \in \mathbb{Z}_q$.

There is an interesting difference between this signature scheme in the symmetric and the asymmetric settings. In the symmetric setting the signer, on being given two valid signatures (A, B, C) and (A', B', C'), is able to tell that they correspond to a randomization of a previous signature, without knowing what that message is. He can do this by verifying that $A' = rA, B' = rB$ and $C' = rC$, for some value r, by performing the following steps:

$$\hat{t}(A', B) = \hat{t}(A, B') \text{ and } \hat{t}(A', C) = \hat{t}(A, C').$$

This makes use of the fact that the DDH problem is easy in $\mathbb{G}_1$ in the symmetric setting.

In the asymmetric setting a signer is unable to determine if two signatures correspond to the same message, since in this setting the DDH problem is believed to be hard in $\mathbb{G}_1$. Indeed one can show that an adversary who can tell whether (A', B', C') is a randomization of (A, B, C), even if the adversary knows x and y, is able to solve DDH in $\mathbb{G}_1$. This difference provides one of the main optimizations of our scheme below.

4 Previous DAA Schemes

In this section we present prior work on DAA schemes, and we analyse their performance.

4.1 Factoring Based Schemes

The original DAA scheme from [7] makes use of the Camenisch-Lysyanskaya signature scheme [10], and hence is based on the strong-RSA assumption. In particular it makes use of a strong-RSA modulus $N = p \cdot q$, i.e. where $p = 2 \cdot p' + 1$ and $q = 2 \cdot q' + 1$ for primes p' and q'. In addition it uses a finite field of prime order Γ. The difficulty of discrete logarithms in $\mathbb{F}_\Gamma$ and of factoring N should be roughly equivalent, so Γ and N are chosen to be roughly the same size.

As in all systems the Setup procedure is rather involved. However, this is only run once and the resulting parameters are only verified once by each party so we ignore the cost of the Setup algorithm and its verification.

In Table 1 table we present the computational cost for all the other algorithms, with respect to each player. An entry of the form

$$1 \cdot \mathbb{G}_N + 2 \cdot \mathbb{G}_\Gamma + 3 \cdot \mathbb{G}_N^2$$

implies that the cost is about one exponentiation modulo N, two modulo Γ and three multiexponentiations with two exponents modulo N, i.e. three operations of the form $g^a \cdot h^b \pmod N$. Note, that a multiexponentiation with m exponents can often be performed significantly faster than m separate exponentiations.

In the table we let P_c denote the cost of generating a prime number of the required size and P_v the cost of verifying that a given number of the required size is prime. We let n denote the number of keys in the verifier's rogue secret key list. We do not specify the time for the linking algorithm, as it is closely related to that of the verification algorithm, and we give the additional time for the RogueTag algorithm over and above the verification algorithm time (which needs to be carried out).

Table 1. Cost of the DAA protocol from [7]

Operation	Party	Cost
Join	TPM	$3 \cdot \mathbb{G}_\Gamma + 2 \cdot \mathbb{G}_N^3$
	Issuer	$n \cdot \mathbb{G}_\Gamma + 2 \cdot \mathbb{G}_N + 1 \cdot \mathbb{G}_N^4 + 1 \cdot \mathbb{G}_\Gamma^2 + P_c$
	Host	$1 \cdot \mathbb{G}_\Gamma + 1 \cdot \mathbb{G}_N^2 + P_v$
Sign	TPM	$3 \cdot \mathbb{G}_\Gamma + 1 \cdot \mathbb{G}_N^3$
	Host	$1 \cdot \mathbb{G}_\Gamma + 1 \cdot \mathbb{G}_N + 1 \cdot \mathbb{G}_N^2 + 2 \cdot \mathbb{G}_N^3 + 1 \cdot \mathbb{G}_N^4$
Verify	Verifier	$4 \cdot \mathbb{G}_\Gamma^2 + 2 \cdot \mathbb{G}_N^4 + 1 \cdot \mathbb{G}_N^6 + n\mathbb{G}_\Gamma$
RogueTag	Verifier	$1 \cdot \mathbb{G}_N^4$

Note that the exponents involved in many of the operations, especially the verification operation, are not of full length. Hence, the above table grossly overestimates the required computational resources. However, one can see that the constrained computing device, namely the TPM is having to perform a considerable number of RSA-length operations.

In [16] a different variant of the DAA protocol is given which tries to reduce the computational cost of the Host, thus allowing trusted computing technologies which use the DAA protocol to be deployed in small devices such as mobile phones. We do not analyse, for reasons of space, the performance of this protocol, but it is also based on factoring assumptions and so all parties need to compute with large integers.

4.2 Symmetric Pairing Based Schemes

Given the increase in RSA key lengths as required by moving to AES key levels, since AES-128 is equivalent to roughly 3000 bits of RSA security, the above two protocols are not going to be suitable in the long term. This led to Brickell et. al. [8] to propose an elliptic curve variant, which reduced the load of the TPM at the expense of requiring pairings to be computed by the other parties.

The Brickell et. al. protocol uses symmetric pairings $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_T$. As above we let $\mathbb{G}_1^m$ etc denote the cost of a multiexponentiation of m values in the group $\mathbb{G}_1$. We also let P denote the cost of a pairing computation. The associated costs are then given by Table 2.

Table 2. Cost of the DAA protocol from [8]

Operation	Party	Cost
Join	TPM	$3 \cdot \mathbb{G}_1$
	Issuer	$(2+n) \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_1^2$
	Host	$6 \cdot P$
Sign	TPM	$4 \cdot \mathbb{G}_T$
	Host	$3 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_T + 3 \cdot P$
Verify	Verifier	$(n+1) \cdot \mathbb{G}_T + 1 \cdot \mathbb{G}_T^2 + 1 \cdot \mathbb{G}_T^3 + 5 \cdot P$
RogueTag	Verifier	$1 \cdot \mathbb{G}_T$

To get some idea of the comparison between the factoring based scheme and the pairing based scheme, consider that the groups $\mathbb{G}_T$ and $\mathbb{G}_N$ (or $\mathbb{G}_\Gamma$) are represented by bit strings of roughly the same size. In addition operations in $\mathbb{G}_T$ can be made slightly more efficient than those in $\mathbb{G}_N$, as in $\mathbb{G}_T$ we can make use of various torus-like representations and tricks, which are not available in standard RSA groups. Finally, the operations in $\mathbb{G}_1$ are about 1/4 the cost of operations in $\mathbb{G}_T$[1].

In the next section we present a variant of the Brickell et. al. pairing based protocol which uses asymmetric pairings. By using asymmetric pairings and

[1] This is a rough estimate derived as follows: $\mathbb{G}_T$ 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 $10/36 \approx 1/4$.

Barreto-Naehrig curves, we are 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 $\mathbb{F}_{q^{12}}$.

5 The Optimized Pairing Based DAA Scheme

We now give a detailed description of the our new DAA scheme based on asymmetric bilinear maps, as opposed to symmetric ones.

5.1 The Setup Algorithm

To set the system up we need to select parameters for each protocol and algorithm used within the DAA scheme well as the long term parameters for each Issuer. On input of the security parameter 1^t the algorithm executes the following:

1. *Generate the Commitment Parameters* $\mathsf{par}_{\mathrm{C}}$. For this three groups, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, of sufficiently large prime order q are selected. Two random generators are selected such that $\mathbb{G}_1 = \langle P_1 \rangle$ and $\mathbb{G}_2 = \langle P_2 \rangle$ along with a pairing $\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. Next a hash function $H_1 : \{0,1\}^* \mapsto \mathbb{Z}_q$ is selected and $\mathsf{par}_{\mathrm{C}}$ is set to be $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{t}, P_1, P_2, q, H_1)$.
2. *Generate the Rogue List Parameters* $\mathsf{par}_{\mathrm{R}}$. A hash function $H_2 : \{0,1\}^* \mapsto \mathbb{Z}_q$ is selected. The rogue list parameters $\mathsf{par}_{\mathrm{R}}$ are then set to be (H_2).
3. *Generate Signature and Verification Parameters* $\mathsf{par}_{\mathrm{S}}$. Two additional hash functions are selected: $H_3 : \{0,1\}^* \mapsto \mathbb{Z}_q$, and $H_4 : \{0,1\}^* \mapsto \mathbb{Z}_q$. We set $\mathsf{par}_{\mathrm{S}}$ to be (H_3, H_4).
4. *Generate the Issuer Parameters* $\mathsf{par}_{\mathrm{I}}$. For each $I_k \in \mathcal{I}$ the following is performed. Two integers are selected $x, y \leftarrow \mathbb{Z}_q$ and the issuer secret key isk_k is assigned to be (x, y). Then the values $X = x \cdot P_2 \in \mathbb{G}_2$ and $Y = y \cdot P_2 \in \mathbb{G}_2$ are computed and the issuer public key ipk_k is assigned to be (X, Y).
   Then an issuer value K_k is computed according to the issuer public values in some predefined manner (we leave the specific details of how this is done as an implementation detail).
   Finally, $\mathsf{par}_{\mathrm{I}}$ is set to be $(\{\mathsf{ipk}_k, \mathrm{K}_k\})$ for each issuer $I_k \in \mathcal{I}$.
5. *Publish Public Parameters.* Finally, the system public parameters par are set to be $(\mathsf{par}_{\mathrm{C}}, \mathsf{par}_{\mathrm{R}}, \mathsf{par}_{\mathrm{S}}, \mathsf{par}_{\mathrm{I}})$ and are published.

The grouping of system parameters is according to usage. For example the set $\mathsf{par}_{\mathrm{C}}$ contains all system parameters necessary for computing commitments and the set $\mathsf{par}_{\mathrm{R}}$ contains those for any rogue checking computations (and also linking).

The group order q is selected so that solving the decisional Diffie–Hellman problem in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ takes time 2^t, as does solving the appropriate bilinear Diffie–Hellman problem with respect to the pairing $\hat{t}$.

An additional optional check of issuer public key values can be added by having each issuer compute $X' = x \cdot P_1$ and $Y' = y \cdot P_1$ and publishing these as part of par. Then to check that both X and Y are correctly formed one simply checks that $\hat{t}(P_1, X) = \hat{t}(X', P_2)$ and $\hat{t}(P_1, Y) = \hat{t}(Y', P_2)$.

5.2 The Join Protocol

This is a protocol between a given TPM $m \in \mathcal{M}$, the corresponding Host $h \in \mathcal{H}$, and an Issuer $I \in \mathcal{I}$. We first give an overview of how a general Join protocol proceeds. There are 3 main stages to a Join protocol. First the TPM m generates some secret message f using the value K_k provided by the issuer and its internal seed DaaSeed. The TPM then computes a commitment on this value and passes this to its Host who adds this to the list of commitments for that user and forwards it to the Issuer. In the second stage the issuer performs some checks on the commitment it receives and, if these correctly verify, computes a credential such that the correctness of this credential can be check by the TPM and Host working together. This credential is passed to the Host in an authenticated manner (using ek). The final stage of a Join protocol involves the Host and TPM working together to verify the correctness of the credential. In our case the Host first performs some computations and stores some values related to these before passing part of the credential on to the TPM prior to verifying the correctness of the credential and then adding this to the list of credentials for that user.

TPM (m)		Host (h)		Initiator (I)
				$n_I \leftarrow \{0,1\}^t$
	$\xleftarrow{\mathsf{comm}_{\mathrm{req}}}$		$\xleftarrow{\mathsf{comm}_{\mathrm{req}}}$	$\mathsf{comm}_{\mathrm{req}} \leftarrow n_I$
$\mathsf{str} \leftarrow 1 \Vert X \Vert Y \Vert n_I$				$\mathsf{str} \leftarrow 1 \Vert X \Vert Y \Vert n_I$
$f \leftarrow H_1(0 \Vert \mathsf{DaaSeed} \Vert \mathrm{K}_k)$				
$u \leftarrow \mathbb{Z}_q$				
$U \leftarrow u \cdot P_1; F \leftarrow f \cdot P_1$				
$c \leftarrow H_1(\mathsf{str} \Vert F \Vert U)$				
$s \leftarrow u + c \cdot f \pmod q$				
$\mathsf{comm} \leftarrow (F, c, s)$	$\xrightarrow{\mathsf{comm}}$		$\xrightarrow{\mathsf{comm}}$	$U' \leftarrow sP_1 - cF$
				If $F = f \cdot P_1$ for some f on the rogue list, or $c \neq H_1(\mathsf{str} \Vert F \Vert U')$ then **abort**
				$r \leftarrow \mathbb{Z}_q$
				$A \leftarrow r \cdot P_1; B \leftarrow y \cdot A$
				$C \leftarrow (x \cdot A + rxy \cdot F)$
				$\mathsf{cre} \leftarrow (A, B, C)$
	$\xleftarrow{\mathcal{E}}$		$\xleftarrow{\mathcal{E}}$	$\mathcal{E} \leftarrow E_{\mathsf{ek}}(\mathsf{cre})$
$\mathsf{cre} \leftarrow E_{\mathsf{ek}}^{-1}(\mathcal{E}); E \leftarrow f \cdot B$	$\xrightarrow{\mathsf{cre}, E}$			
		$\rho_a \leftarrow \hat{t}(A, X)$		
		$\rho_b \leftarrow \hat{t}(B, X)$		
		$\rho_c \leftarrow \hat{t}(C, P_2)$		
		If $\hat{t}(A, Y) \neq \hat{t}(B, P_2)$ or $\hat{t}(A + E, X) \neq \rho_c$ then **abort**		

Fig. 1. The Join Protocol

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

- If the points P_1, P_2, X, Y are not formed correctly then this could leak information about the value of a given f, for example due to small subgroup attacks. To prevent this from happening each TPM needs to verify that P_1 generates $\mathbb{G}_1$, P_2 generates $\mathbb{G}_2$ and that $X, Y \in \mathbb{G}_2$. This need be done once for each TPM so we do not give this as part of the Join protocol. Algorithms for checking whether points are elements of particular pairing groups are given in [14].
- The value of cre is not sent in the clear and hence only the intended user can obtain the complete credential. This is done by encrypting the value cre under a public key corresponding to the TPM endorsement key ek. Again, we do not consider these calculations in the performance analysis of the scheme.
- In contrast with the RSA-based DAA schemes 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, on the value F computed using the secret key f.
- Once a credential is issued from I, the TPM and the Host verify that this credential is correctly formed. This is to avoid performing computations with a credential that is incorrectly formed since this could lead to leaking information about the value f held by the TPM. The last part of the protocol therefore performs the verification algorithm from the Camenisch-Lysyanskaya signature scheme. In addition the TPM should check that the value of B it receives in the credential is correctly formed. Since $B \in \mathbb{G}_1$ this can be performed very efficiently and so we ignore its cost when computing the cost of running the protocol.
- We note that the Host does not perform any verification on values that are provided by the TPM. Since we assume that it is harder to compromise a TPM than a Host, we do not model the case of a corrupt TPM inside an honest Host and hence the Host will always trust the correctness of values provided by its TPM.
- The values ρ_a, ρ_b, ρ_c and E are stored for later use by the Host in the signing algorithm. This improves the performance by avoiding recomputation of various pairing values.

5.3 The Sign Protocol

This is a protocol run between a given TPM $m \in \mathcal{M}$ and Host $h \in \mathcal{H}$. The objective of the sign protocol is for m and h to work together to produce a signature of knowledge on some message. The signature should prove knowledge of a discrete logarithm f, knowledge of a valid credential and that this credential was computed for the same value f. 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 internal value of f published (i.e. it is not a rogue module) then the Host will not know f and will be unable to compute the whole signature without the aid of the TPM.

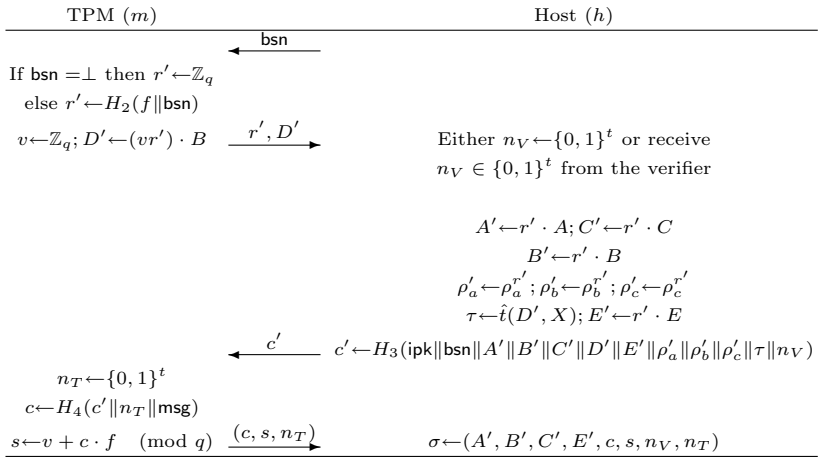

Fig. 2. The Sign Protocol

We again assume that we could have an adversarially controlled Host and honest TPM and, as a result, the TPM will have to do a number of checks on the data passed to it from its Host. We let msg denote the message to be signed and bsn denote the base name of the verifier. The protocol then proceeds as in Figure 2, so as to produce the signature σ.

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

- 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 offline manner we allow the Host to generate its own value of n_V.
- 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 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 other signatures future signatures to be able to be link to this one then it chooses a verifier bsn that it has not used before.
- The use of E' allows the verifier to identify if the signature was produced by a rogue TPM by computing $f_i \cdot B'$ for all f_i values on the rogue list and comparing these to E'. This is performed during the verification algorithm. Without E' the rogue test algorithm can be performed by using elements in $\mathbb{G}_T$, however in practice this is much less efficient than using E'.
- During the run of the signature protocol two nonces are used: one from the verifier n_V and one from the TPM n_T. These are used to ensure each

signature is different from previous signatures and to ensure that 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.
– The value r' is used to mask the signature created from the other players in the scheme including the issuer. Without using r' the credential on which the signature is computed would be sent in the clear and hence other parties would be able to link signatures. That the issuer cannot link the signatures follows from the earlier mentioned property of the Camenisch-Lysyanskaya signature scheme in the asymmetric setting. Thus it provides two types of linking resistance: It stops any issuer from being able to link a given signature to a given signer (since issuers know the values of r used to compute a credential and without r' the credential is sent in the clear), and it stops any player in the system from being able to tell if any two signatures were produced by the same signer (if different bsn are used).
Note, that 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.

5.4 The Verification Algorithm

This is an algorithm run by a verifier 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 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 a signature σ of the form $\sigma = (A', B', C', E', c, s, n_V, n_T)$ this algorithm performs the following steps:

1. *Check Against Rogue List.* If $E' = f_i \cdot B'$ for any f_i in the set of rogue secret keys then return *reject*.
2. *Check Correctness of A' and B'.* If $\hat{t}(A', Y) \neq \hat{t}(B', P_2)$ then return *reject*.
3. *Verify Correctness of Proofs.* This is done by performing the following sets of computations:
   – $\rho_a^\dagger \leftarrow \hat{t}(A', X)$, $\rho_b^\dagger \leftarrow \hat{t}(B', X)$ and $\rho_c^\dagger \leftarrow \hat{t}(C', P_2)$.
   – $\tau^\dagger \leftarrow (\rho_b^\dagger)^s \cdot (\rho_c^\dagger / \rho_a^\dagger)^{-c}$
   – $D^\dagger \leftarrow sB' - cE'$.
   – $c^\dagger \leftarrow H_3(\mathsf{ipk} \| \mathsf{bsn} \| A' \| B' \| C' \| D^\dagger \| E' \| \rho_a^\dagger \| \rho_b^\dagger \| \rho_c^\dagger \| \tau^\dagger \| n_V)$.
   Finally if $c \neq H_4\left(c^\dagger \| n_T \| \mathsf{msg}\right)$ return *reject* and otherwise return *accept*.

5.5 The Linking Algorithm

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

TPM. 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 pair of signatures σ_b for $b \in \{0, 1\}$ each having the form $\sigma_b = (A'_b, B'_b, C'_b, E'_b, c_b, s_b, n_{V,b}, n_{T,b})$ the algorithm performs the following steps:

1. *Verify Both Signatures.* For each signature σ_b the verifier runs $\mathsf{Verify}(\sigma_b)$ and if either of these returns *reject* then the value $\perp$ is returned.
2. *Compare Signatures and Basenames.* If the two basenames which verify the signatures are equal, and if $A'_0 = A'_1$ then return *linked*, else return *unlinked*.

It may be the case that one or both signatures input to the 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.

Note, our linking algorithm works due to the way that r' is computed in the signing algorithm. Also note that anyone who knows bsn can link the two signatures, but they cannot link the signatures with the signers.

5.6 The Rogue Tagging Algorithm

The purpose of the rogue tagging algorithm is to ensure that an adversary is not able to tag a given value of TPM internal secret as rogue if the TPM that owns that particular value is not corrupted.

On input a value of f and a signature σ intended for a given verifier V_j the algorithm then proceeds as follows:

1. *Verify the Signature.* If $\mathsf{Verify}(\sigma) = \textit{reject}$ then the value $\perp$ is returned.
2. *Check Value of* f. If $E' \neq f \cdot B'$ then return $\perp$ and otherwise add an entry f to $\mathsf{RogueList}(V_j)$.

We note that, since the credential computed for a given user is sent using a secure channel, the only way that an adversary can produce a valid signature would be if it knew the value of the credential and hence had corrupted that user to some extent. This prevents the adversary from adding arbitrary values of f to $\mathsf{RogueList}(V_j)$.

5.7 Efficiency Comparison

Table 3 presents the performance analysis of our optimized version of the pairing based DAA protocol. We use a similar notation for computational cost as in our previous tables. The main advantages of our version can be listed as follows:

- Due to DDH being hard in $\mathbb{G}_1$ we can remove a number of the checks and masks in the original pairing based DAA protocol.
- In addition we move the computation of τ in the signature from the TPM to the Host. This removes the need for the TPM to perform any $\mathbb{G}_T$ operations at all.

– The Host precomputes some pairing values at the Join stage so as to remove the need to perform these at the signing stage. This comes at the expense of a couple more $\mathbb{G}_T$ operations. But an exponentiation in $\mathbb{G}_T$ is cheaper than a pairing.

Table 3. Cost of our DAA protocol

Operation	Party	Cost
Join	TPM	$3 \cdot \mathbb{G}_1$
	Issuer	$(2+n) \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_1^2$
	Host	$6 \cdot P$
Sign	TPM	$1 \cdot \mathbb{G}_1$
	Host	$4 \cdot \mathbb{G}_1 + 3 \cdot \mathbb{G}_T + 1 \cdot P$
Verify	Verifier	$n\mathbb{G}_1 + 1 \cdot \mathbb{G}_1^2 + 1 \cdot \mathbb{G}_T^2 + 5 \cdot P$
RogueTag	Verifier	$1 \cdot \mathbb{G}_1$

The main point to note is that the TPM is only required to perform operations in $\mathbb{G}_1$, which can be an elliptic curve over a relatively small finite field. Thus the TPM does not have to perform any expensive operations at all.

In conclusion we have presented a DAA protocol based on pairings for which the TPM, i.e. the constrained device in the system, needs very little computational resources in comparison to other variants of the DAA protocol. This efficiency has been achieved by moving to the asymmetric pairings setting and by various precomputations. Our protocol can be proved secure in the random oracle model under the strongest of the two security notions in the literature for DAA schemes. The security proof will be in the full version of the paper.

References

1. 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.
2. I.F. Blake, G. Seroussi and N.P. Smart. *Elliptic curves and cryptography.* Cambridge University Press, 1999.
3. D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology - Eurocrypt 2004*, Springer-Verlag LNCS 3027, 223–238, 2004.
4. D. Boneh, X. Boyen and H. Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, Springer-Verlag LNCS 3152, 41–55, 2004.
5. D. Boneh, G. Di Crescenzo, R. Ostrovsky and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology – Eurocrypt 2004*, Springer-Verlag LNCS 3027, 506–522, 2004.

6. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 213–229, 2001.
7. E. Brickell, J. Camenisch and L. Chen. Direct anonymous attestation. *Proceedings of the 11th ACM Conference on Computer and Communications Security*. ACM Press, 132–145, 2004.
8. E. Brickell, L. Chen and J. Li. Simplified security notions for direct anonymous attestation and a concrete scheme from pairings. *Cryptology ePrint Archive*. Report 2008/104, available at `http://eprint.iacr.org/2008/104`.
9. E. Brickell, L. Chen and J. Li. A new direct anonymous attestation scheme from bilinear maps. To appear *Proceedings Trust 2008*.
10. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communications Networks – SCN 2002*, Springer-Verlag LNCS 2576, 268–289, 2003.
11. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO 2004*, Springer-Verlag LNCS 3152, 56–72, 2004.
12. J. Camenisch and M. Michels. A group signature scheme based on an RSA-variant. Technical Report RS-98-27, BRICS, University of Aarhus, 1998.
13. L. Chen, Z. Cheng, J. Malone-Lee and N.P. Smart. An efficient ID-KEM based on the Sakai-Kasahara key construction. *IEE Proceedings, Information Security*, **153**, 19–26, 2006.
14. L. Chen, Z. Cheng and N.P. Smart. Identity-based key agreement protocols from pairings. *Int. Journal of Information Security*, **6**, 213–242, 2007.
15. S. Galbraith, K. Paterson and N.P. Smart. Pairings for cryptographers. To appear, 2008.
16. H. Ge and S.R. Tate. A Direct Anonymous Attestation Scheme for Embedded Devices. *Proceedings of Public Key Cryptography – PKC 2007*. Springer-Verlag LNCS 4450, 2007.
17. 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.
18. F. Hess, N.P. Smart and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, **52**, 4595–4602, 2 006.
19. S. Mitsunari, R. Sakai and M. Kasahara. A new traitor tracing. *IEICE Transactions on Fundamentals*, **E85-A(2)**, 481–484, 2002.
20. A. J. Menezes, T. Okamoto and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inf. Theory*, **39**, 1639–1646, 1993.
21. Trusted Computing Group. `www.trustedcomputinggroup.org`.