On a Possible Privacy Flaw in Direct Anonymous Attestation (DAA)

Adrian Leung[1]*, Liqun Chen[2], and Chris J. Mitchell[1]

[1] Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
{A.Leung,C.Mitchell}@rhul.ac.uk
[2] Hewlett-Packard Laboratories
Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK
Liqun.Chen@hp.com

Abstract. A possible privacy flaw in the TCG implementation of the Direct Anonymous Attestation (DAA) protocol has recently been discovered by Rudolph. This flaw allows a DAA Issuer to covertly include identifying information within DAA Certificates, enabling a colluding DAA Issuer and one or more verifiers to link and uniquely identify users, compromising user privacy and thereby invalidating the key feature provided by DAA. In this paper we argue that, in typical usage scenarios, the weakness identified by Rudolph is not likely to lead to a feasible attack; specifically we argue that the attack is only likely to be feasible if honest DAA signers and verifiers never check the behaviour of issuers. We also suggest possible ways of avoiding the threat posed by Rudolph's observation.

Keywords: Direct Anonymous Attestation, DAA, Privacy, Trusted Computing.

1 Introduction

Direct Anonymous Attestation (DAA), proposed by Brickell, Camenisch and Chen, [1, 2], is a special type of group signature scheme that can be used to anonymously authenticate a principal, also referred to as a prover, to a remote verifier. DAA has been adopted by the Trusted Computing Group[3] in version 1.2 of the Trusted Computing Trusted Platform Module (TPM) Specifications [3]. The key features provided by DAA are the capability for a prover (a group member) to anonymously convince a remote verifier that:

- it is in possession of a DAA Certificate obtained from a specific DAA Issuer, without having to reveal the DAA Certificate to a verifier (as would be necessary for a signature-based proof of knowledge);

* This author is supported by the British Chevening/Royal Holloway Scholarship, and in part by the European Commission under contract IST-2002-507932 (ECRYPT).

[3] http://www.trustedcomputinggroup.org/

- a DAA Signature computed by a prover on a message m, has been generated using a valid DAA Certificate issued by a specific DAA Issuer; colluding verifiers are unable to link two different DAA Signatures created by the same prover, and, in particular, verifiers are not given the DAA Certificate.

Moreover, the DAA scheme provides a flexible way of achieving a number of different levels of 'linkability'. Under an agreement between the prover and verifier, DAA Signatures can be either 'random-base' or 'name-base'. Two random-base signatures signed by the same prover (TPM) for the same verifier cannot be linked. However, name-base signatures are associated with the verifier's name; as a result, two name-base signatures signed by the same prover (TPM) for the same verifier can be linked.

These features help to protect the privacy of a prover. Another important feature of DAA (distinguishing it from other types of group or ring signature schemes) is that the powers of the supporting Trusted Third Party (i.e. the DAA Issuer in its role as the group manager) are minimised, as it cannot link the actions of users (i.e. provers) and hence compromise the user's privacy even when it colludes with a verifier. This unlinkability property is the key feature of DAA.

However, an attack was recently discovered by Rudolph [4] which potentially compromises the unlinkability property of DAA. The attack could allow a DAA Issuer to embed covert identifying information into DAA Certificates (of provers) and to subsequently link the transactions of the users/provers to whom the DAA Certificates belong [4]. As a result, DAA Signatures originating from the same users/provers become linkable, and users can thereby be uniquely identified. In this paper, we argue that Rudolph's attack may be infeasible in practice, and we discuss why an attempt to launch such an attack could easily be discovered in an environment where there is at least one honest verifier. We also propose approaches which could prevent the attack from taking place.

The remainder of this paper is organised as follows. In Section 2 we briefly describe the workings of DAA as well as outlining the privacy attack. In Section 3 we explain why the attack is likely to be unrealistic in many practical scenarios, and, in Section 4, we discuss possible modifications to the use of DAA in the TCG specifications that can prevent the attack. Finally, conclusions are drawn in Section 5.

2 The Privacy Attack on DAA

In this section, we only describe those aspects of the DAA protocol necessary to understand the Rudolph attack. For full details of DAA, including a proof of its security, see [1] and chapter 5 of [2]. A brief description of the attack then follows.

2.1 DAA Overview

We first introduce the entities involved in the DAA protocol and the roles they play.

- The *Certification Authority (CA)* acts as a Trusted Third Party (TTP). Its role is to certify the authenticity of the DAA Issuer's longer-term public key, CK_I. It does not directly participate in the DAA protocol.
- The *DAA Issuer* (or just the Issuer) is a third party that issues DAA Certificates (i.e. anonymous credentials) to provers. It must be trusted by all other participants in the DAA protocol to perform its role in a trustworthy manner.
- The *Prover* (or the User) generates DAA Signatures that are verified by verifiers. In the context of Trusted Computing, the prover is the TPM.
- The *Verifier* verifies DAA Signatures computed by provers.

Note that, apart from the CA, all the entities above take direct part in the DAA protocol. Also, in normal circumstances, the numbers of CAs and DAA Issuers are likely to be very small by comparison with the number of provers.

The DAA Scheme consists of two sub-protocols (or phases), namely the *DAA Join Protocol* and the *DAA Sign Protocol*. In the Join Protocol (shown in Figure 1), a prover interacts with a DAA Issuer I in order to obtain an anonymous credential on a secret value f (also referred to as the DAA Secret), known only to the prover. This anonymous credential, also known as the DAA Certificate, is jointly computed by the DAA Issuer and the prover as a function of a blinded value of f, the shorter-term public key of I, PK_I, and other parameters. The DAA Certificate is later used by a prover during the DAA Sign Phase in the DAA Signature computation. As part of the DAA Join protocol, the prover is authenticated to the DAA Issuer using its unique and long lived Endorsement Key (EK). Note that the public part of the EK can be used to uniquely identify a prover. The Issuer authenticates itself to the prover using its shorter-term public key PK_I, which the prover verifies using a certificate signed by the Issuer with its longer-term public key CK_I, which is in turn certified by the CA.

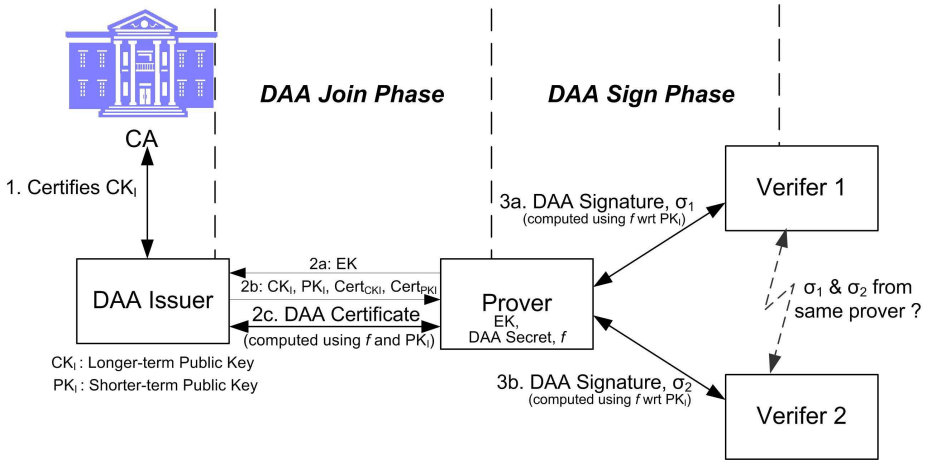

Fig. 1. The DAA Scheme

In the DAA Sign Phase, the prover DAA-signs a message m, using its DAA Secret f, the DAA Certificate, and other public parameters. The output of the DAA-signing process is known as the DAA Signature σ. This DAA Signature enables the prover to prove to a verifier (using a signature-based proof of knowledge) that (i) it is in possession of a DAA Certificate, and (ii) the DAA Signature on message m was computed using its DAA Secret f, the DAA Certificate in (i), and other public parameters. Verifying a DAA Signature requires knowledge of the DAA Issuer's public key PK_I (i.e. the public key of the DAA Issuer that was used to create the DAA Certificate). Hence prior to running the DAA Sign protocol, a verifier must have obtained an authentic copy of PK_I.

The DAA Sign Protocol has the property that colluding verifiers are unable to link different DAA Signatures originating from the same prover (as shown in Figure 1). This property applies even if a DAA Issuer colludes with a verifier, i.e. they are still unable to link and uniquely identify a particular prover.

2.2 The Rudolph attack

The privacy breaching attack on DAA proposed by Rudolph [4] operates under an assumption about the use of DAA, which we first describe. Specifically, Rudolph assumes that the DAA Issuer's longer-term public key CK_I, as well as the (shorter-term) public key PK_I (along with its certificate chain) used to compute the DAA Certificate, are communicated to the verifier via the prover during the DAA Sign Phase (as shown in Figure 2). Whether or not this is a realistic assumption is not clear; other possibilities include use of a publicly accessible certificate repository. In any event, as we describe below, the verifier will need to know which of the DAA Issuer's shorter-term keys has been used to create the DAA Certificate on which the DAA Signature is based.

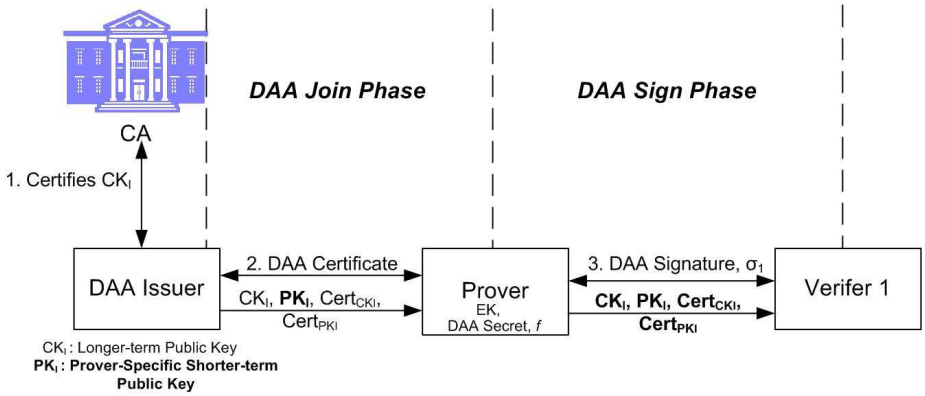

Fig. 2. The Rudolph Attack

The attack works as follows. As shown in Figure 1, during the DAA Join Phase the DAA Certificate is computed using the DAA Issuer's public key PK_I and other parameters. The key PK_I is a shorter-term public key which is certified using the longer-term public key CK_I, which in turn is certified by a trusted CA (as shown in Figure 1). This key hierarchy is an intentional design feature of DAA, chosen to make the TPM's key life cycle and the Issuer's key life cycle independent. This is because the TPM computes its DAA private key as a digest of a secret seed and the Issuer's longer-term public key. This ensures that the TPM uses the Issuer key that matches its DAA private key. If the Issuer had only a single key, then, when the Issuer changed its key, every TPM would also have to update its key. To avoid this problem, the Issuer is given the two types of key described above.

Unfortunately, this flexibility can potentially be exploited by a curious DAA Issuer to compromise the privacy properties of DAA. As observed by Rudolph, a DAA Issuer could embed covert identifying information into a public key without the knowledge of an honest prover. The Issuer simply uses a different public key PK_I to generate DAA Certificates for each prover with which it interacts. As a result, the Issuer will be able to compile a table mapping between a prover's public EK (its unique key) and the public key used to generate the DAA Certificate for this prover.

Suppose a verifer has obtained the Issuer's public key PK_I from the prover. Whenever a prover executes the DAA Sign protocol with a colluding verifier (i.e. one that colludes with the Issuer to identify a prover), and assuming that the public key PK_I used to generate its DAA Certificate is communicated to a verifier via the prover, the DAA Issuer and the colluding verifier can easily link the transactions of a prover and uniquely identify it. A verifier needs only inform the DAA Issuer of the value of PK_I. The DAA Issuer can then consult the EK-PK_I mapping it has compiled, and determine the prover's EK. Similarly, with the aid of the Issuer, colluding verifiers are able to uniquely identify a particular prover.

3 How realistic is the Rudolph attack?

We now consider how the Rudolph attack might work in practice, and in particular we examine two possible attack scenarios.

3.1 Scenario 1: Linking large numbers of users

In this scenario, the aim of the DAA Issuer is to identify large numbers of provers. We believe that this attack scenario is infeasible in practice. We demonstrate (i) why this attack scenario is unrealistic (even if all the verifiers collude) because of the communications and computation burden involved in performing such an attack; and (ii) how the attack can easily be detected if there is at least one honest verifier:

(i) The Rudolph attack only works if the DAA Issuer and all the verifiers collude. Suppose we have a scenario where there is one DAA Issuer, n provers (all joining the network or system at different times), and k verifiers (which all collude with the Issuer in an attempt to link or uniquely identify the provers). For the attack to work, the colluding verifiers have to shoulder an additional communication and computational burden (see Table 1 for a summary of the communication and computational overheads). First and foremost, to be able to link all the n provers, the DAA Issuer would need to use n different public keys PK_I, one for each prover. These n public keys would also need to be communicated to each of the colluding verifiers. If a trusted directory is used to hold copies of the public keys, then the Issuer would need to upload a total of n different public keys to this directory (if the provers join at different times then this may involve sending up to n separate upload messages). If the verifiers obtain the public keys from the Issuer directly, then the total communications overhead for an Issuer may be up to nk messages (as compared to k messages if the Issuer only uses one key).

A colluding verifier, regardless of the mechanism used to retrieve Issuer public keys, will need to obtain up to n Issuer public keys for the n provers. This means that the communications overhead for a verifier may be up to n messages. Furthermore, whenever there is a new prover, the verifiers might need to obtain the new public key for this prover.

In terms of computational overheads, we now point out why it is by no means trivial to launch the Rudolph attack. Firstly, the generation of the DAA Issuer public key PK_I involves performing a non-interactive zero knowledge proof (using the Fiat-Shamir heuristic) [1]. This potentially involves the DAA Issuer performing at least 160 modular exponentiations (which could go up to 6×160, one for each of the public key components g, h, S, Z, R_0 and R_1). This contradicts Rudolph's claim that the process of generating a large number of public keys can be performed efficiently [4].

Secondly, the work to be performed by the verifier in trying to identify the prover may become infeasibly large, depending on how Issuer public keys are distributed. If the prover sends the Issuer-signed certificate for the Issuer public key PK_I to the verifier as part of the DAA Sign protocol, as assumed by Rudolph, then there is no problem. However, if the key is obtained from a directory, then significant computational problems arise. This is because the verifier will have no way of knowing which of the n Issuer public keys have been used to create the DAA certificate, and hence which of them should be used to attempt to verify the DAA Signature. The only solution would be for the verifier to attempt to verify the signature using every possible key, which would involve up to n verifications. Given the ubiquity of trusted computing hardware, a typical value of n might be 10^5 or 10^6, which would make such a process computationally infeasible.

(ii) The attack will easily be discovered if there is at least one honest verifier or if Issuer public keys are stored in public directories, as we now describe.

Table 1. Communications and computation costs for honest and colluding entities.

Type of Costs	Costs Incurred By			
	Honest Issuer	Honest Verifier	Colluding Issuer	Colluding Verifier
Communication (no. of messages)	1	1	nk	n
Computation (no. of DAA Sign Verifications)	-	1	-	n

- Consider first an environment in which there is at least one honest verifier. When the honest verifier attempts to verify a DAA Signature, it first needs to retrieve the Issuer's public key from the Issuer (or from a trusted directory). If the Issuer (or the trusted directory) submits a large number of public keys to the verifier, then suspicions about its trustworthiness will immediately be aroused. Even if the honest verifier is given the Issuer public key by the prover rather than retrieving it from a directory, then it could still detect misbehaviour if it keeps a log of all the Issuer public keys that it has been passed. If one particular Issuer is using large numbers of different public keys, then this will quickly become obvious to such a verifier.
- Suppose an Issuer uploads multiple public keys to a directory or other repository. This will immediately be obvious both to the operator of the directory (which may report suspicious behaviour) as well as to any user of the directory.

3.2 Scenario 2: Linking a small set of users

On the other hand, if the aim of the DAA Issuer is to link all transactions involving a single user (e.g. a high-profile user or one that makes high value transactions), or a very small set of users, then the attack is much more likely to succeed in a way that is hard to detect. For example, if a DAA Issuer only wants to distinguish transactions involving one user, then the Issuer only needs to have two public keys PK_I. Hence, the communication and computation problems discussed in the previous section would not be an issue in this attack scenario, nor would there be a large number of Issuer public keys in circulation to arouse suspicions.

Nevertheless, if a verifier deals with many provers who are clients of the same Issuer, the suspicions of an honest verifier might be aroused if one Issuer public key, or some small set of such keys, is used much less than others. In particular, if the DAA Signatures are of the name-base type, allowing a verifier to link

DAA Signatures signed by the same prover (TPM) for the same verifier, then the verifier will be able to observe significant differences in the numbers of clients for an Issuer's public keys.

4 Preventing the Rudolph attack

Despite the issues raised in the previous section, the Rudolph attack will work if a verifier obtains the Issuer's public key from a prover (as described in [4]), and if no honest verifier keeps track of the Issuer public keys it has received or if the goal of the attackers is only to track a few users. Even worse, a prover would be completely oblivious to such an attack, as there is no way for a prover to tell if a DAA Issuer is embedding covert identity information into the public key that is used to generate its DAA Certificate (e.g. by using a different public key for each prover).

We now examine a number of possible ways of preventing the Rudolph attack. We also discuss the limitations of these approaches.

4.1 Modifying the TCG Specifications

We first observe that addressing the root cause of the problem would involve changing the TCG specifications to prevent a DAA Issuer from self-certifying an arbitrary number of public keys. This could be achieved by requiring the Issuer to use a private key for which the public key has been directly certified by a third party CA (in the notation used above, this would mean that the issuer key CK_I would be used to generate DAA certificates).

There are two problems with such a approach. Firstly, the CA would then need to be trusted not to generate large numbers of certificates for an Issuer. Whilst this could be supported by requiring any CA that generates Issuer certificates to adhere to an appropriate Certification Practice Statement, it still means that a significant amount of trust is placed in a single third party, a situation which the design of DAA seeks to avoid.

Secondly, as explained above, this means that the key life cycle of the TPM and the Issuer become linked. Addressing this issue would require further changes in the operation of the TPM.

An alternative approach would retain the two levels of Issuer public keys, but would require both types to be certified by a CA. As in the existing scheme, the first level key would be used to compute the DAA secret, and the second level key would be used to create the DAA Certificate. Certificates for second level keys could mention the first level key to link the two together. This could address the Rudolph attack without causing a key life cycle issue.

4.2 Using a Trusted Auditor

We next explore another possible approach which does not involve modifying the TCG specifications too much (or at all). We propose that a prover obtains

DAA Certificates only from DAA Issuers that use the same public key for a very large set of users. Thus the challenge is to enable a prover to determine the key usage behaviour of a DAA Issuer.

If a prover is able to obtain assurance that a specific Issuer's public key has being used more than a certain number of times (i.e. to generate a certain number of DAA Certificates), then it is immediately able to derive confidence that it will not be uniquely identified, and at the same time gain information about the level of anonymity that it is being afforded. For example, if a prover knows that the public key used to generate its DAA Certificate has been used to generate a thousand other DAA Certificates, then it knows that it cannot be distinguished from a thousand other entities. On the other hand, if it knows that a particular public key has only been used to generate three other DAA Certificates, then the level of anonymity afforded to it is potentially very limited.

We now suggest two possible approaches designed to give a prover this type of assurance. We also discuss the possible limitations of the suggested approaches.

A modification to the use of DAA. This approach requires the introduction of a new type of trusted third party, which we refer to as a *Trusted Auditor* (TA). We make use of the TA (which is not necessarily unique) to give provers assertions about the "trustworthiness" of individual DAA Issuers. Since the CA needs to be trusted by the protocol participants, and since it is already employed to certify the longer-term public key CK_I of the Issuer, a CA could act as a TA, although this does need to be the case.

We propose that the following additional steps be performed by a prover during the DAA Join protocol. During the Join phase, and after a DAA Certificate has been successfully created, the prover establishes a secure channel with the TA. This can be achieved by the prover first establishing a unilaterally authenticated secure channel using a public key certificate for the TA, e.g. using SSL/TLS. The prover can then send its EK credential to the TA via this channel, and use this credential to authenticate itself, e.g. by decrypting data sent to it encrypted using the EK. Finally, the prover uses this channel to send a statement that a specific DAA Issuer has used a particular public key PK_I to derive its DAA Certificate, i.e.

$$Prover \rightarrow TA : ID_{Issuer}, CK_I, PK_I$$

Using these messages, the TA can compile a list of the form given in Table 2. A copy of this list signed by the TA (to prove authenticity) can then be communicated to a prover prior to the Join Phase. Since it is desirable not to publicise the public EK values of individual trusted platforms, this information can be removed from the list before it is distributed. The list of public EKs can be replaced by the total number of different EKs for which credentials have been generated using a particular Issuer public key.

This approach suffers from one problem. The public part of the Endorsement Key (EK) of every prover is revealed to the TA. Since the EK pair for a trusted platform is fixed, the public EK functions as a fixed identifier for a platform,

Table 2. Mapping of EKs to a PK_I for individual Issuers

No.	Issuer Name	Longer-term CK_I	Shorter-term PK_I	Users
1.	Alice	CK_I^A	PK_I^{A1}	EK_1 ⋮ EK_{2000}
			PK_I^{A2}	EK_1 ⋮ EK_{10}
2.	Bob	CK_I^B	PK_I^{B1}	EK_1 ⋮ EK_{200}

and hence revealing it is not desirable. Indeed, DAA was introduced to avoid the need to reveal the link between the public EK and other prover public keys to a Privacy CA. Nevertheless, this scheme does not pose the same threat to user privacy as the use of a Privacy CA, since the TA does not learn the link between the EK and any other prover keys.

An alternative approach. A possible alternative to the above approach avoids revealing the EK to a third party, although it still relies on a TA to provide assertions regarding the trustworthiness of the DAA Issuer (i.e. reporting on the number of DAA Certificates generated for a particular public key for a specific Issuer). A prover and DAA Issuer run the DAA Join protocol as normal, thereby obtaining a DAA Certificate (for the prover's unique DAA Secret f). The prover then conducts an instance of the DAA Sign protocol with the TA, which acts as the verifier. The prover DAA-signs the following information sent to the TA (acting as verifier), so that the TA can compile a table similar to that shown in Figure 2.

$$Prover \rightarrow TA : ID_{Issuer}, CK_I, PK_I$$

One possible problem with this approach is that, because no use is made of the EK, a malicious Issuer could fabricate DAA-signed messages of the above form, and send them to the TA. The DAA signature signed for the TA could be of the name-base type, which will guarantee that each DAA secret can only provide one piece of evidence. However, the messages could be based on any number of 'fake' DAA Certificates, that are valid in that they have been created by the DAA Issuer, but have never been sent to a genuine prover. Such messages will be indistinguishable from messages sent from genuine provers, and hence the number of uses of a key can be artificially inflated.

Nevertheless, a table created in this way will still reveal if an Issuer has created more public keys than would be expected in 'normal' behaviour; this

may be sufficient to deter an Issuer from using the Rudolph attack on a large scale.

4.3 A User-Centric Approach

To determine the trustworthiness of an Issuer, two or more users could collaborate to compare the PK_I values that they have obtained from a particular Issuer. If all of them have the same key PK_I, then there is good chance that the Issuer is using the same PK_I for a large set of users. However, if the users find that two or more different keys PK_I have been used, then the trustworthiness of the issuer is immediately called into question. This approach is suited to a distributed or peer-to-peer environment, and does not require the involvement of a trusted third party. Clearly, the effectiveness of the technique will depend on the number of cooperating users.

5 Concluding Remarks

A privacy flaw in DAA was recently pointed out by Rudolph [4]. In this paper we have analysed the feasibility of attacks exploiting this property. We then examined possible approaches which could be used to prevent (or reveal) the attack as well as the limitations of these approaches. These approaches could make a successful attack very difficult to perform; however, all of the suggestions have certain drawbacks. It remains an open problem to find a 'perfect' solution to the Rudolph attack. Indeed, the DAA scheme itself cannot stop an issuer from using a different key with each TPM, no matter whether the key is certified by the Issuer's longer-term key or by another CA. It is a very tough challenge for any application to completely avoid such a threat.

It should be noted that protocols and applications (such as those given in [5–7]) which employ DAA as a building block are not affected by this attack, as it is not an attack on the DAA protocol itself, but is rather a weakness introduced in the particular use of DAA. In fact, an implementation of an arbitrary group signature scheme might have a similar problem if the size of a group is very small, e.g. for groups containing just a single member. However, in other implementations of group signatures, a group manager might not have any motivation to break the anonymity of its members, because the manager has the ability to open the identity of a signer from its signature.

Acknowledgements

We thank Heiko Stamer and Stéphane Lo-Presti for their valuable comments, and in particular to Heiko Stamer for pointing out the true complexity of generating DAA Issuer key pairs. We also thank Rene Mayrhofer for bringing the Rudolph attack to our attention.

References

1. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington DC, USA, October 25–29, 2004, ACM Press (2004) 132–145
2. Mitchell, C.J., ed.: Trusted Computing. IEE Press, London (2005)
3. Trusted Computing Group (TCG): TCG Specification Architecture Overview. Version 1.2, The Trusted Computing Group, Portland, Oregon, USA (2004)
4. Rudolph, C.: Covert identity information in direct anonymous attestation (DAA). In Venter, H., Eloff, M., Labuschagne, L., Eloff, J., von Solms, R., eds.: 22nd IFIP TC-11 International Information Security Conference (SEC2007) on "New Approaches for Security, Privacy and Trust in Complex Environments", Sandton, South Africa, May 14-16, 2007. Proceedings. Volume 232 of IFIP International Federation for Information Processing., Springer, Boston (2007) 443–448
5. Balfe, S., Lakhani, A.D., Paterson, K.G.: Trusted computing: Providing security for peer-to-peer networks. In: Proceedings of the Fifth International Conference on Peer-to-Peer Computing (P2P'05), Konstanz, Germany, August 31–September 2, 2005, IEEE Computer Society (2005) 117–124
6. Leung, A., Mitchell, C.J.: Ninja: Non identity based, privacy preserving authentication for ubiquitous environments. In Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T., eds.: 9th International Conference on Ubiquitous Computing (Ubicomp 2007), Innsbruck, Austria, September 16–19, 2007. Proceedings. Volume 4717 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2007) 73–90
7. Leung, A., Poh, G.S.: An anonymous watermarking scheme for content distribution protection using trusted computing. In: Proceedings of the International Conference on Security and Cryptography (SECRYPT 2007), Barcelona, Spain, August 28–31, 2007, INSTICC Press (2007) 319–326