RZ 3590 (# 99600) 01/31/05
Computer Science 21 pages

Research Report

Protecting (Anonymous) Credentials with the Trusted Computing Groups Trusted Platform Modules V1.2

Jan Camenisch

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
E-mail: jca@zurich.ibm.com

Protecting (Anonymous) Credentials with the Trusted Computing Group's Trusted Platform Modules V1.2

Jan Camenisch

IBM Research, Zurich Research Lab
Säumerstrasse 4, CH-8803 Rschlikon, Switzerland
jca@zurich.ibm.com

November 8, 2004

Abstract

Digital credentials and certificates can easily be shared and copied. For instance if a user possesses a credential that allows her to access some service, she can easily share the credentials with her friends and thereby enable her friend to access the service as well. While with non-anonymous credentials, this sharing can to some extend be detected by the fact that some credentials get used too often, such detection is not possible with anonymous credentials. Furthermore, the honest user is also at risk of identity theft: malicious software such as viruses and worms or phishing attacks can without too much difficulty steal her credentials.

One solution to the problem is to use tamper-resistant hardware tokens to which a credential is bound such that a credential can only be used in connection with the token. Although this approach is sometimes taken for isolated high security applications, it is not used widely because of the organizational overhead to distribute such tokens. Moreover such tokens are usually very application specific and hence cannot be used with different applications (from different service providers).

Recently, however, manufacturers have started to embed into computers a tamper-resistant piece of hardware, called trusted platform modules (TPM), as specified by the Trusted Computing Group. In this paper we show that this module can in fact be used to secure anonymous as well as non-anonymous credentials. That is, we provide a mechanism to insure that credentials can only be used with the TPM it got issued to. We then extend our solution to one that allows the use of credentials not only with the TPM they got issued to but also with other TPMs of the *same* user. Finally, we show how to secure a full-fledged anonymous credential system. Once TPMs are widely distributed, our solution can offer for the first time strong and privacy friendly authentication for electronic transactions.

1 Introduction

Due to their nature, digital credentials can easily be copied and distributed. Thus, on the one hand, a computer virus or worm, for instance, can easily steal a user's credentials and, on the other hand, a user can even easier share her credentials (illegitimately) with her friends. In case credentials are used to protect access to valuable resources or services, such things should obviously be prevented and the credentials themselves be protected as well.

Although one can obtain some protection by software (such as storing credentials only in encrypted form and trying to secure the operating system), containment of credentials in a hardware token offers much better protection. The idea here is that the credentials (or at least their secret parts) never leave the hardware token and are only processed inside the token and can only be used in a well specified manner. While such protection is in principle possible also for credentials consisting of username and password (e.g., using so-called password-based key exchange protocols [2, 27, 34, 24]), one would rather use public key cryptography because passwords have low entropy and are really targeted

towards credentials that need to be stored in a human brain. Credentials based on public key cryptography work as follows: A public/private key pair is generated (on the hardware token) and the public key is sent to the issuer of the credential. The issuer either just stores the public key in a list of authenticated keys or actually issues a certificate on the public key, i.e., signs the public key together with some further (access) information or attributes. When a user then wants to use her credentials to access some service or resource, she sends the public key (and possibly the certificate) to the resource or service provider. The provider then either checks whether the public key is in the list of authorized keys that is made available by the issuer or verifies the validity of certificate on the public key. Finally, using the secret key, the user (or the hardware token) identifies as the owner of the public key, and then will obtain access to the requested service or resource. If one uses a so-called zero-knowledge protocol for this identification process (e.g., [22, 30]), this process does not divulge the secret key. Thus, if the only interface a hardware token offers is to generate a key pair, output the public key, and to run a zero-knowledge identification protocol w.r.t. this key pair, no information about the secret key is leaked from the hardware token and the credential cannot be used without the hardware token. Thus, the credential is protected from malicious software such as viruses and worms. Furthermore, assuming that the hardware token is tamper-resistant and that the issuer assures that the secret key is indeed held in the hardware token, users can no longer share credentials without sharing the token as well.

We note that this kind of protection is in principle also applicable to so-called anonymous credentials [16, 8]. Such credentials work basically in the same way except that the user's transaction with the issuer and the one with the service/resource provider cannot be linked. That is, even if the user if fully identified by the issuer, the service provider learns only that he is communicating with a user who got authorized by the issuer to access the service/resource. Thus anonymous credentials allow the protection of the users' privacy and are in fact one of the main ingredients to implement privacy principles in electronic transactions.

Today, credentials are only rarely protected by hardware tokens. One reason for this is that the cost of the deployment of such tokens is still too high, in particular as the tokens are not interoperable and can hence only be used for isolated applications. In fact, weak authentication in combination with the growing rate of frauds such as phishing attacks seriously hinder the growth of e-commerce on the Internet and even turn people away from using the Internet for financial transactions.

TCG's Trusted Platform Module. Recently, manufacturers have started to embed so-called trusted platform modules (TPM) [31, 32] into computers. These modules are one of the building blocks for trusted computing that are specified by the Trusted Computing Group (TCG) [33]. They are essentially smartcard chips build into a computing device. These modules can, among other things, generate public and secret key pairs. Moreover, they have a mechanism to remotely convince another party that a key pair was indeed generated by them and that the secret part of it is protected by the module (i.e., cannot not be extracted from the module except by physically breaking it open).

More precisely, upon manufacturing each of these modules generates an encryption key pair, called Endorsement Key (EK) in the TCG language, the public key of which get certified by the manufacturer as being one of a valid TPM. At a later time (e.g., upon request of the user), the TPM can generate a so-called attestation identity key (AIK) which is an RSA [29] signing key pair and can thus also be used for identification. To convince a remote party that the secret portion of an AIK is indeed held within a valid TPM, two mechanisms are specified by the TCG. The first one often referred to as the "Privacy CA" method and is as follows. The TPM (via the device it is embedded into) sends the AIK public key together with the EK to a certification authority (called privacy CA in the TCG language). The certification authority checks whether the EK was certified by the manufacturer, and if so, issues a certificate on the AIK public key, encrypts this certificate under the EK, and sends this encryption back to the TPM. The TPM decrypts this and, if it is indeed a certificate on an AIK it generates, output the certificate to the device/user. As the TPM will not output certificates on AIK it has not generated, the certification authority, or any other party who trust the certification authority, will, upon seeing the certificate, be convinced that the AIK secret key is indeed held inside an valid TPM.

However, this solution has the drawback the Privacy CA can link the EK to AIK generated by TPM and therefore

is not too privacy friendly. Moreover, the availability of such a trusted party is a strong assumption. Indeed, TCG was criticized by consumer groups as well as by data-protection officials, which lead TCG to develop and specify a second, more privacy friendly solution called direct anonymous attestation (DAA) [32, 5]. This new protocols draws on techniques that have been developed for group signatures [18, 13, 1], identity escrow [26], and credential systems [16, 8]. Such a signature allows members of the group to anonymously sign on behalf of the group, i.e., the verifier of a group signature is not able to tell which particular group member originated the signature. In fact, direct anonymous attestation can be seen as a group signature scheme without the capability to open signatures (or anonymity revocation) but with a mechanism to detect rogue members (TPMs in this case). Thus, upon manufacturing, the TPM is joined the group of valid TPM by the manufacturer. Later, the TPM can generate an AIK and authenticate it with DAA, i.e., by using a group signature to sign the AIK.

Our Contribution. In prinicple one could use the TPM to secure credentials by having the TPM generate an AIK key pair, proving that the key pair was indeed generated by the TPM (using either the Privacy CA solution or DAA), and then issue a certificate on the AIK public key. However, as pointed out earlier, this approach does not protect the user's privacy Moreover, does not extend to anonymous credentials.

In this paper we show how the TPM's functionalities can be used to secure credentials in a privacy friendly way. That is, we analyse the DAA protocol and show how the TPM's part of that protocol can be used to obtain a new, DAA-like protocol that allows one to issue (anonymous) credentials to users in such a way that they can only be used in conjunction with the TPM they got issued to. This protocol provides anonymity to users, i.e., the issuer of the credential and the provider of the service or resource cannot link the two transactions. Essentially, we obtain an anonymous credential system where a user can only use her credentials in cooperation with her TPM. We note that our new protocol does not require any modification of the TPM as specified by the TCG [32] and hence our solution can be realized with the TPMs that will be found in computing devices starting in 2005.

While our solution offers the users protection against viruses and the service providers protection against fraudulent users, it has the drawback that a user can use a credential only with the device (resp., the TPM embedded into this device) the credential was issued to. Thus, credentials are not portable as they would for instance be if they were tied to a smartcard or a USB key. We therefore extend our solution to allow a user to use all *her* credentials with all *her* devices. We also show how to protect conventional (i.e., non-anonymous) credentials as well how to extend our scheme to a full-fledged anonymous credential system.

We believe that once TPMs according to the V1.2 specification [32] become widely available, our solution can enable widespread strong authentication and privacy protection for electronic transactions and help ignite the staggering growth of e-commerce on the Internet.

2 Model and Security Requirements of Our Scheme

A system for protecting anonymous credentials for use with all of a user's devices and *only* with those is a follows. It consists of the following parties: a device-credential issuer, an application-credential issuer, a service (or resource) provider, a number of devices, and a number of users. While the model could easily be extended to include several issuers and providers, it is sufficiently to consider only a single instance of them. The system features the following procedure.

KeyGenDevCredI and KeyGenAppCredA. These are the key generators for the device-credential and the application credential issuer. On input a security parameter λ, they output a public/secret key pair (PK_D, SK_D), resp., (PK_A, SK_A), for the device-credential and the application credential issuer, respectively.

GetDevCred. This is a protocol between a user's device and a device-credential issuer. The input to both parties are the latter's public key PK_D and the user's identity ID_U. The issuer's additional input is his secret signing key SK_D. In case the user is eligible to register the device (e.g., because she has not yet registered too many

devices), the user's device's output of the protocol is a device credential and some related secret keys (which will be held inside the device's TPM).

GetAppCred. This is a protocol between a user's device and an application-credential issuer. The input to both parties are the latter's public key PK_A. The user's device gets as additional input the identity ID_U. The application-credential issuer's additional input is his secret signing key SK_A. In case the user is eligible to obtain an application credential, the user's device's output of the protocol is an application credential and some related secret keys, all of which the device outputs.

UseAppCred. This is a protocol between a user's device and a service provider. The input to both parties are the public keys PK_A and PK_D of the device- and application-credential issuers. The device's input its device credential and the related secret keys (the latter are kept in the device's TPM), the user's identity, as well as an application credential the user has obtained with one of her devices and the related secret keys. The output of the service or resource provider is either accept or reject.

The registration of a user's device does not necessarily require the user's identity – it can in principle also be done anonymously. We discuss this issue in §4.2, but for now require the user to identify herself to register a device.

The security requirements are as follows:

Unforgeability. We require that no adversary who can register as many devices and retrieve as many application credentials as he wishes, can successfully run the protocol UseAppCred with a service-provider with an unregistered device or can successfully run the protocol UseAppCred with more different (but hidden) identities that he has obtained an application credentials.

Anonymity. We require that even if the device-credential issuer, the application-credential, and the service provider collude, they cannot link different transactions by the same user except those transactions to register devices that were conducted under the same identity.

3 Preliminaries

3.1 Commitment Schemes

A first building block for our scheme, is a *statistically hiding commitment scheme* Consider a domain X. A commitment scheme to elements in X is given by a family $\{\mathsf{Comit}\}_{n\in\mathbb{N}}$, where $\mathsf{Comit}_n : X \times \{0,1\}^{r(n)} \to \{0,1\}^{l(n)}$; here $r(n)$ represents the number of random coins used to commit, and $l(n)$ is the bit-length of such a commitment. The security requirement that we need is that the scheme is statistically hiding, i.e., for any $x_0, x_1 \in X$, the distribution ensembles $\{\mathsf{Comit}(x_0, U(r(n))\}_n$ and $\{\mathsf{Comit}(x_1, U(r(n))\}_n$ are statistically indistinguishable, where $U(r(n))$ denotes the random variable of choosing an integer uniformly from $\{0,1\}^{r(n)}$. We are using essentially the scheme of [23, 20]: if $\mathfrak{G}$ is a group of unknown order (for example $\mathbb{Z}_\mathfrak{n}$ with $\mathfrak{n}$ an RSA modulus with unknown factorization,) and $\mathfrak{g}$ and $\mathfrak{h}$ are random group elements such that $\mathfrak{g} \in \langle\mathfrak{h}\rangle$ then $\mathsf{Comit}(x)$ is defined by $\mathfrak{g}^x\mathfrak{h}^r \bmod \mathfrak{n}$, where r is randomly chosen from a big enough domain. The scheme is computationally binding if the committing party is not privy to the factorization of $\mathfrak{n}$.

3.2 Proof Protocols for Discrete Logarithm related Statements

In our scheme we will use various protocols to prove knowledge of and relations among discrete logarithms. In particular, it is known that the following things can be proven.

1. Proof of knowledge of a discrete logarithm modulo a prime [30] or a composite [23, 21].

2. Proof of knowledge of equality of representation modulo two (possibly different) prime [17] or composite [11] moduli.
3. Proof that a commitment opens to the product of two other committed values [10, 14, 4].
4. Proof that a committed value lies in a given integer interval [15, 10, 10, 3].
5. Proof of the disjunction or conjunction of two of the above [19].

To describe these protocols, we use notation introduced by Camenisch and Stadler [13] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For instance, $PK\{(\alpha,\beta,\gamma) : y = g^\alpha h^\beta \ \wedge \ \tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma \ \wedge \ (u \leq \alpha \leq v)\}$ denotes a "*zero-knowledge* P*roof of* K*nowledge of integers* α*,* β*, and* γ *such that* $y = g^\alpha h^\beta$ *and* $\tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma$ *holds, where* $u \leq \alpha \leq v$*,*" where $y, g, h, \tilde{y}, \tilde{g}$, and $\tilde{h}$ are elements of some groups $G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. The convention is that Greek letters denote the quantities the knowledge of which is being proved, while all other parameters are known to the verifier. Using this notation, a proof protocol can be described by just pointing out its aim while hiding all details.

In the random oracle model, such protocols can be turned into signature schemes using the Fiat-Shamir heuristic [22, 28]. We use the notation $SPK\{(\alpha) : y = g^\alpha\}(m)$ to denote a signature obtained in this way and call it proof signature.

3.3 The Camenisch-Lysyanskaya Signature Scheme.

The direct anonymous attestation scheme as well as our new scheme are both based on the Camenisch-Lysyanskaya (CL) signature scheme [9]. Unlike most signature schemes, this one is particularly suited for our purposes as it allows for efficient protocols to prove knowledge of a signature and to retrieve signatures on secret messages efficiently using discrete logarithm based proofs of knowledge [9]. As we will use somewhat different (and also optimized) protocols for these tasks than those provided in [9], we recall the signature scheme here and give an overview of discrete logarithm based proofs of knowledge in the following subsection.

Key generation. On input 1^k, choose an RSA modulus $n = pq$ being a safe-prime product, i.e., with $p = 2p' + 1$, $q = 2q' + 1$. Choose, uniformly at random, $R_0, \ldots, R_{L-1}, S, Z \in \mathrm{QR}_n$. Output the public key $(n, R_0, \ldots, R_{L-1}, S, Z)$ and the secret key p. Let ℓ_n be the length of n.

Signing algorithm. Let ℓ_m be a parameter. On input $(m_0, \ldots, m_{L-1})$ with $m_i \in \pm\{0,1\}^{\ell_m}\}$, choose a random prime number e of length $\ell_e > \ell_m + 2$, and a random number v of length $\ell_v = \ell_n + \ell_m + \ell_r$, where ℓ_r is a security parameter. Compute the value A such that $Z \equiv R_0^{m_0} \ldots R_{L-1}^{m_{L-1}} S^v A^e \pmod{n}$. The signature on the message $(m_0, \ldots, m_{L-1})$ consists of (e, A, v).

Verification algorithm. To verify that the tuple (e, A, v) is a signature on message $(m_0, \ldots, m_{L-1})$, check that $Z \equiv A^e R_0^{m_0} \ldots R_{L-1}^{m_{L-1}} S^v \pmod{n}$, and check that $2^{\ell_e} > e > 2^{\ell_e - 1}$.

Theorem 1 ([9]). *The signature scheme is secure against adaptive chosen message attacks [25] under the strong RSA assumption.*

The original scheme considered messages in the interval $[0, 2^{\ell_m} - 1]$. Here, however, we allow messages from $[-2^{\ell_m} + 1, 2^{\ell_m} - 1]$. The only consequence of this is that we need to require that $\ell_e > \ell_m + 2$ holds instead of $\ell_e > \ell_m + 1$.

3.4 Anonymous Credentials with Camenisch-Lysyanskaya Signatures

Camenisch and Lysyanskaya [9] also presented secure protocols to 1) prove knowledge of a signature on committed messages and to 2) get a signature on committed messages such that the signer does not learn the messages. We

recall these protocols. In fact, we present somewhat more efficient protocols incorporating improvements presented by Brickell, Camenisch, and Chen [5] and by Camenisch and Groth [7].

Let $(n, R_0, \ldots, R_{L-1}, S, Z)$ be the public key of the signer such that $R_0, \ldots, R_{L-1}, Z \in \langle S \rangle$ is assured (cf. [5] for the latter).

3.4.1 Obtaining a Signature

A signature on messages $m_0, \ldots, m_{L-1} \pm \{0,1\}^{\ell_m}$, where the messages $m_0, \ldots m_{L'-1}$, $L' \leq L$, shall be hidden from and the messages $m_{L'}, \ldots m_{L-1}$ known to the signer $\mathcal{S}$, can be obtained by a signature receiver $\mathcal{R}$ as follows.

1. $\mathcal{R}$ chooses a random $v' \in_R \{0,1\}^{\ell_n + \ell_\varnothing}$, computes $U := S^{v'} \prod_{i=0}^{L'-1} R_i^{m_i} \mod n$ and sends U to signer.

2. $\mathcal{R}$ executes as prover to verifier $\mathcal{S}$ the following protocol

$$PK\{(m_0, \ldots, m_{L'-1}, v') : U \equiv \pm S^{v'} \prod_{i=0}^{L'-1} R_i^{m_i} \pmod n \wedge m_0, \ldots, m_{L'-1} \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_{\mathcal{H}} + 2} \wedge v' \in \{0,1\}^{\ell_n + \ell_\varnothing + \ell_{\mathcal{H}} + 2}\} \ .$$

3. $\mathcal{S}$ chooses $\hat{v} \in_R \{0,1\}^{\ell_v - 1}$ and a prime $e \in_R [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$, computes $v'' := \hat{v} + 2^{\ell_v - 1}$ and

$$A := \Big(\frac{Z}{U S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^{1/e} \mod n \ ,$$

   and sends (A, e, v'') to $\mathcal{R}$.

4. To convince $\mathcal{R}$ that A was correctly computed, $\mathcal{S}$ as prover runs the protocol

$$PK\{(d) : \quad A \equiv \pm \Big(\frac{Z}{U S^{v''} R_2^{k_t}}\Big)^d \pmod n\}$$

   with $\mathcal{R}$ as verifier.

5. $\mathcal{R}$ checks whether e is a prime and lies in $[2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$. $\mathcal{R}$ stores $(A, e, v := v' + v'')$ as signature on the messages $m_0, \ldots m_{L-1}$.

Note that $\mathcal{R}$ is not privy of the factorization of n and $R_0, \ldots, R_{L-1} \in \langle S \rangle$ and hence the value U is a statistically hiding and computationally binding commitment to the messages $m_0, \ldots m_{L'-1}$ (cf. §3.1).

The proof in Step 2 will convince $\mathcal{S}$ that $\mathcal{R}$ "knows" the messages committed by U and thus the values computed in Step 3 will indeed be a valid CL-signature and not just reveil an e-th root of some value $\mathcal{R}$ concucted. Finally, the proof in Step 4 will convince $\mathcal{R}$ that $\mathcal{S}$ compute the value A correctly, in particular that $A \in \langle S \rangle$. The latter is important to guarantee privacy when proving possession of a signature with the protocol in the next section.

3.4.2 Proving Possession of a Signature

To prove possession (or knowledge) of a signature on a message, the following protocol can be use. We demonstrate the protocol here for the case where the messages m_i with $i \in I_r$ are revealed to the verifier, the messages m_i with $i \in I_c$ are hidden from the verifier but of which she receives a commitment, and finally no information whatsoever about the messages m_i with $i \in I_h$ is revealed to the verifier. Assume there are available $\mathfrak{g}$, $\mathfrak{h}$, and $\mathfrak{n}$ to be used for a commitment scheme as described in §3.1. (The values $\mathfrak{g}$, $\mathfrak{h}$, and $\mathfrak{n}$ could for instance be chosen by the verifier or by some trusted third party.)

1. For $i \in I_c$, $\mathcal{R}$ chooses $r_i \in \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $C_i := \mathfrak{g}^{m_i}\mathfrak{h}^{r_i} \bmod \mathfrak{n}$, and sends C_i to $\mathcal{V}$ and $\{m_i | i \in I_r\}$
2. $\mathcal{R}$ chooses $r \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $A' := AS^r$, and sends A' to $\mathcal{V}$.
3. Finally, $\mathcal{R}$ as prover executes the protocol

$$\begin{aligned} PK\{(e, \tilde{v}, \{m_i | i \in I_c \cup I_h\}) : \\ Z \prod_{i \in I_r} R_i^{-m_i} \equiv \pm A'^e S^{\tilde{v}} \prod_{i \in I_c \cup I_h} R_i^{m_i} \pmod{n} \ \wedge \ C_i \equiv \mathfrak{g}^{m_i}\mathfrak{h}^{r_i} \pmod{\mathfrak{n}} \ (i \in I_c) \wedge \\ m_i \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_{\mathcal{H}}+2} \ (i \in I_c \cup I_h) \ \wedge \ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_{\mathcal{H}}+1}\} \end{aligned}$$

   with $\mathcal{V}$ as verifier.
4. The verifier also checks that $m_i \in \pm\{0,1\}^{\ell_m}$ for $i \in I_r$.

Note that $(A', e, v + r)$ is also a valid signature on the messages $m_0, \ldots m_{L-1}$.

Let us discuss the reasons why we allow for some messages to be hidden or only given as commitments. The reason for this it to allow the protocol to be used in higher level applications. The messages can for instance be used to encode attributes into a credentials. Examples of such attribute include which kind of resource the user is allows to access, the user's identity, the expiration date of the certificate, etc. Now, depending on the application and to protect the user's privacy, not all of these attributes should be revealed. For instance, the verifier only sees the message encoding the kind of resource the credential allows the user to access, but only get commitments to the expiration date and possibly also to the identity. Using these commitments, the user can then run a separate proof with the verifier showing that, for instance, the credential not expired yet, i.e., that the expiration date is bigger than the current date or, if required by the application, the user could also provide the verifier with an verifiable encryption [6, 12] of the contents of the commitment to her identity under a trusted third party's public key. The latter will allow the trusted third party to revoke the user's anonymity, .e.g., in case she misuses the anonymous access to the resource.

3.5 Direct Anonymous Attestation

The direct anonymous attestation (DAA) protocol [5, 32] applies the Camenisch-Lysyanskaya signature and the protocols above to issue a certificate (attestation) to a computing platform that it is genuine. More precisely, the attestation is issued to the trusted platform module (TPM) embedded into the platform. As we aim to use the TPM to bind any credential to it, let us describe in more details how the direct anonymous attestation protocol works.

The involved parties are an attester, a platform, a TPM embedded in the platform, and a verifier that wants to get convinced by the platform that it got attestation. The idea is that the TPM chooses two secret "messages" m_0 and m_1, obtains a Camenisch-Lysyanskaya (CL) signature (aka attestation) on it from the attester the protocol described in the previous section. The reason that the TPM chooses two instead of just one secret message is technical: it allows the TPM for a larger probability space for the secret while allowing for a smaller message size which results in smaller (and therefore more efficiently chosen) primes e in the certificates. When the platform wants to prove to verifier that it has embedded attested TPM, the TPM generates a pseudonym N_V using (m_0, m_1) and then proves that it has gotten a signature from the attester on the messages committed by N_V. This proof is done as described in the previous section with the only difference that N_V is a special commitment, i.e., $N_V = \zeta^{m_0+2^\ell m_1}$ for some given ζ and ℓ. We refer to [5] for more details on N_V.

As the TPM is a small chip, the direct anonymous attestation protocol was designed such that all operations for retrieving a signature and proving possession of a signature that could be outsourced to the platform and computed there. In particular, as a platform could always destroy the privacy of its TPM, all operations that are necessary for privacy but not for security are performed by the platform.

For the protocol to obtain a signature, the TPM basically only performs $\mathcal{R}$'s part of the Steps 1 and 2, i.e., chooses m_0, m_1, and v', computes $U := S^{v'} R_0^{m_0} R_1^{m_1} \pmod n$, and the proof of knowledge of correctness of U. $\mathcal{R}$'s part Steps 3 and 4 are executed by the platform (except that the TPM computes $v := v' + v''$, as v (and v') needs to be kept secret).

For the protocol to prove possession of a signature (attestation), the TPM computes the "commitment" N_V and the verifier part of the proof protocol

$$\mathit{PK}\{(m_0, m_1, v) : \quad U \equiv R_0^{m_0} R_1^{m_1} S^v \pmod n \ \wedge N_V \equiv \zeta^{m_0 + f_m 2^\ell} \ \wedge \ m_0, m_1 \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H} + 2}\} \ .$$

Note that this proof reveals U, i.e., does not hide the TPM's (and thus the platform's) identity. The platform then transform the TPM's messages of this proof, using its knowledge of A and e, into messages of the following proof (c.f. §3.4)

$$\mathit{SPK}\{(m_0, m_1, \tilde{v}, e) : \quad Z \ \equiv \pm A'^e S^{\tilde{v}} R_0^{m_0} R_1^{m_1} \pmod n \ \wedge \ N_V \equiv \zeta^{m_0 + f_m 2^\ell} \ \wedge \\ m_0, m_1 \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H} + 2} \ \wedge \ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_\mathcal{H} + 1}\} \ ,$$

where $A' := AS^r$ for a randomly chosen r (cf. §3.4).

4 Securing Credentials with a TPM

This section describes how one can use the parts of the DAA protocol executed by the TPM to issue a credential that are tied to a TPM in such a way that the credential can only be used if one has access to that TPM. To this end, we are going to show how the DAA protocol can be extended the cover the same functionality as the protocol presented in §3.4.1, i.e., such that the issuer (attester) can sign not only the (secret) messages m_0 and m_1 hidden from him but many messages. Of course, the messages m_0 and m_1, i.e., the TPM's secret keys, remain hidden inside the TPM (this is ensured by the DAA protocol, resp. by the operations that the TPM performs). We will then show how the platform can prove (in cooperation with the TPM) that it has obtained such a signature. While these two protocols are merely a stepping stone to our final solution, they already allow one to secure anonymous credentials by tying them to TPMs: As credentials that are tied in this way to a TPM can no longer be used without the cooperation of the TPM, the user is protected from malicious code such as viruses. Also a credential issuer is protected from malicious users because such credentials can not longer be shared by different users (or at least only be users of the same platform). However, as TPMs are build into particular devices, this is not very convenient for users who use a number of devices (e.g., laptop, PDA, mobile phone, home-computer, office-computer, etc) and want to use their credentials with all their devices without retrieving each credential with each of these platforms. That is users would like to have their credentials portable as would for instance be the case if the credentials were tied to a smartcard. To overcome this drawback of credentials that are tied to a TPM, we present a scheme that allows a user to transfer credentials between her devices and use all of her credentials with all of them.

In the description of the protocols, we use the terms TPM, device, and platform as follows. TPM is the module built into the device and will execute some operations on its own. To communicate with the outside world, the TPM is dependent on the other components of the device. We call all these other components the platform. That it, we consider devices to consist of two (separate) parties, a TPM and a platform, where the TPM communicates only with the platform.

4.1 Extending DAA-Credentials to Include Attributes

If one inspects the direct anonymous attestation protocol [32, 5], one find that the way the operations between the TPM and the platform are shared, allows one to extend the protocol to cover the whole spectrum of obtaining and using

credentials offered by protocols described in §3.4. We describe these extensions in the following on a conceptional level. The detailed protocols are provided in Appendix A.

We stress that these extensions do not require any modification of the TPM (which is a piece of hardware) as specified in [32], but only modifications of the platform part of DAA which is all software (cf. [5]).

We note that in the DAA scheme [5], all the interactive proof protocols PK are executed in their non-interactive form SPK obtained via the Fiat-Shamir heuristic (cf. §3.2). We therefore also stick to this in the remainder of the paper.

4.1.1 Obtaining a Credential That is Tied to a TPM

To obtain a credential on messages $m_2, \ldots, m_{L-1} \in \pm\{0,1\}^{\ell_m}$ that is tied to a TPM, we extend the DAA protocol such that the signature the issuer generates is not only a signature on the hidden messages m_0 and m_1 (which are the secret keys chosen by the TPM) but also a signature on the messages $m_2, \ldots, m_{L-1} \in \pm\{0,1\}^{\ell_m}$. The idea is that the TPM, the platform, and the issuer execute the following step which we here describe only at a high level. The detailed protocol that shows exactly what operations the TPM, the platform and the issuer perform are provided in Appendix A.1.

1. While the TPM computes U (and some other value N_I), the platform chooses a random $\hat{v} \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $U' := S^{\hat{v}} \prod_{i=2}^{L'-1} R_i^{m_i} \bmod n$ and sends to the signer the value U' together with the U produced by the TPM. Furthermore, the value U is authenticated by the TPM using its endorsement key EK. We refer to [5, 32] for the details on how this is achieved.

2. While the TPM will act as prover in the protocol that U and N_I is correctly formed, the platform adds the parts to the proof that U' is correctly formed, i.e., they together produce the verifier parts of the proof protocol

$$\begin{aligned} \textit{SPK}\{(m_0, \ldots, m_{L'-1}, v', \hat{v}) : U \equiv \pm S^{v'} R_0^{m_0} R_1^{m_1} \pmod{n} \ \wedge\ N_I := \zeta_I^{m_0 + m_1 2^{\ell_m}} \pmod{\Gamma} \wedge \\ U' \equiv \pm S^{\hat{v}} \prod_{i=2}^{L'-1} R_i^{m_i} \pmod{n} \wedge \\ m_0, \ldots, m_{L'-1} \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_{\mathcal{H}}+2} \ \wedge\ v' \in \{0,1\}^{\ell_n+\ell_\varnothing+\ell_{\mathcal{H}}+2}\} \ . \end{aligned}$$

   This protocol convinces that signer that the TPM has computed U correctly and the platform did build U' correctly. (Note that U and U' are essentially commitments to the messages m_i).

3. Now the signature is generated by the signer as in the DAA protocol (and as in the protocol in§3.4.1) with the exception that A is computed as

$$A := \Big(\frac{Z}{UU'S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^{1/e} \bmod n \ ,$$

   where v'' and e are the two random values chosen by the issuer to compute A. That is, the signer here uses two commitments U and U' to the messages instead of only one to compute A. The signer also sends the values (A, e, v'') to the platform.

4. The signer's proof that A is correctly formed is done in the same way as in the DAA protocol (and as in the protocol in§3.4.1) with the adaption to the two values U and U', i.e., the proof protocol becomes

$$\textit{SPK}\{(d) : \quad A \equiv \pm\Big(\frac{Z}{UU'S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^d \pmod{n}\} \ .$$

5. This step is also unchanged, apart that the platform needs to store more things now, i.e., the TPM stores $m_0, m_1, v = v'' + v'$ and the platform stores $A, e, \hat{v}, m_2, \ldots, m_L$.

4.1.2 Proving Possession of a Credential that is tied to a TPM

To show possession of an extended DAA-certificate, i.e., a signature by the DAA-issuer that now is not only a signature on the TPM's secret keys m_0 and m_1 but also on the messages $m_2, \ldots, m_{L-1} \in \pm\{0,1\}^{\ell_m}$, we have to modify the DAA-sign protocol as follows. Again, the TPM's part of the DAA protocol remains the same. However, the platform now needs to extend the TPM's prover's messages to prover's messages of a proof that includes all terms about the messages $m_2, \ldots, m_{L-1}$. That is the platform receives from the TPM, which performs the steps of the original DAA protocol, the prover's messages for the proof protocol

$$\mathit{SPK}\{(m_0, m_1, v): \quad U \equiv R_0^{m_0} R_1^{m_1} S^v \pmod{n} \ \wedge N_V \equiv \zeta^{m_0 + f_m 2^\ell} \ \wedge \ m_0, m_1 \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H} + 2}\}$$

and into the prover's messages of the protocol

$$\begin{aligned}\mathit{PK}\{(e, \tilde{v}, \{m_i | i \in I_c \cup I_h\}): \\ Z \prod_{i \in I_r} R_i^{-m_i} \equiv \pm A'^e S^{\tilde{v}} \prod_{i \in I_c \cup I_h} R_i^{m_i} \pmod{n} \wedge \equiv \mathfrak{g}^{m_i} \mathfrak{h}^{r_i} \pmod{\mathfrak{n}} \ (i \in I_c) \wedge \\ m_i \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H} + 2} \ (i \in I_c \cup I_h) \ \wedge \ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_\mathcal{H} + 1} \ ,\end{aligned}$$

where, as described in §3.4, the value A' is derived from A as $A' = AS^r$ for a suitable chosen r, the values C_i are commitments to messages, and I_c and I_h are set of indices of the messages to which the verifier receives the commitments C_i and which remain hidden from the verifier, respectively.

The full protocol for is provided in Appendix A.2.

4.2 Using Credentials with Several Devices

We have seen how to tie a Camenisch-Lysyanskaya signature [9] to a TPM such that possession of a signature (i.e., credential) can only be proved in cooperation with that particular TPM. As mentioned, this prevents users from (legitimately) using their credentials with several of their platforms as they could to with (ordinary) credentials that are not tied to a piece of hardware or to a (re-)movable hardware token such as a smartcard. We now present a scheme that allows users to use their credentials with all of their device while still ensuring that a user's credential cannot be used without one of that user's devices.

The idea is that the user registers (the TPM's of) all her devices with some authority. That is, the registration authority issues the user's TPM an extended DAA-credential that contains as one of the signed messages (e.g., m_2) a identifier that is unique to that user such as her identity. To this end, the authority and the user's device run the protocol we have described in §4.1.1. We call such a credential an device credential. The "real" credentials, i.e., those that will allows the user to access some resource or service (we call them application credential in the following) are then issued to the user as described in §3.4 except that we always set one of the message, e.g., m_0, to the user's identity. Note that these credentials are not directly tied to the TPM. In case the user's identity should not get known to the issuer of application credentials, the user can provide him with a commitment C to m_0, use the protocol presented in §4.1.2 to show that she has obtained a device-credential that contains as message m_2 (which is her identifier) that is committed to by C, and then the protocol in §3.4 to obtain a signature on a committed message. Now, whenever a user want to use an application credential, we require the user to provide a commitment C the her identifier and then to prove that she has obtained a device credential that contains as second message the value committed to by C and the she has further obtained an application credential that contains as 0-th message also the value committed to by C. This she can do using the protocols provided in §4.1.2 and §3.4.2, respectively.

It is not hard to see that this solution provides what we want: Application credentials cannot be used without having access to a device that got registered w.r.t. the identity that is contained in the application credential. Thus, if the device credential issuer makes sure that only a limited number of devices get registered under the same identity, application credentials can only be used with a limited number of devices. In fact, it is not important that the string

encoded into both the device credentials and the application credential is the user's identifier. The device-credential issuer only needs to make sure that only a limited amount of devices get registered per identifier. Depending on the scenarios, one might have different limits here. Thus, the registration of a device could even be pseudonymous. That is the first time a user registers a device he would establish a pseudonym with the device credential issuer who would then choose an identifier for the user to be encoded into the device credential.

Depending on the scenario and the limits on how many devices a user can register, one might nevertheless want to ensure that users with few devices register their devices w.r.t. the same identifier. In the non-pseudonymous case, one could achieve this be requiring that the user fully identifies herself. In the pseudonymous case, one would require that the user prove ownership of her pseudonym (of course only after she has registered the first device) and apply one of the known methods to prevent the user from sharing her pseudonym (cf. [8]). One such a method is for instance to bind the secret key required to prove ownership of the pseudonym to, e.g., the user's bank account. Thus, if the user would share her pseudonym, she would also share her bank account. The details of all of this, however, are out of the scope of this paper and we assume for simplicity in the following description of our solution that each user has a unique identifier.

4.2.1 Setup

For simplicity we consider only a single issuer of device credentials and a single issuer of application credentials. However, the scheme is easy extendable to several registration authorities. Let $(n, R_0, \ldots, R_{L-1}, S, Z)$ be the former's public key for the CL-signature scheme and $(\mathsf{n}, \mathsf{R}_0, \ldots, \mathsf{R}_{L-1}, \mathsf{S}, \mathsf{Z})$ be the one of the latter. That is the key generation algorithms $\mathsf{KeyGenDevCredI}$ and $\mathsf{KeyGenAppCredA}$ are the key generation algorithms of the CL-signature scheme.

4.2.2 Registering a Device

The procedure $\mathsf{GetAppCred}$ to register a device is as follows. The user's device runs the protocol to obtain a signature that is tied to a TPM as described in §4.1 with the parameters $L' = 2$ and $L = 3$ and $m_2 = ID_U$. Thus, the user's device will obtain values (A, e, ID_U) such that

$$Z \equiv A^e R_0^{m_0} R_1^{m_1} R_2^{ID_U} S^v \pmod{n}$$

holds, where the values m_0, m_1, and v are known only to the TPM of the device the user just has registered.

4.2.3 Obtaining a Credential That is Tied to All of the User's TPMs

We now describe the procedure $\mathsf{GetAppCred}$.

To obtain an application credential, the user could in principle just run the protocol to obtain a signature provided in §3.4.1 where $m_0 = ID_U$ would be message hidden from the (application credential) issuer. I.e., there is per se not need for the application issuer to verifier that the user has obtained a device credential because later on the user won't be able to use the obtained application credential if she has not obtained a device credential containing the same identifier as the one contained in the application credential.

As a result of this protocol $(\mathsf{A}, \mathsf{e}, \mathsf{v})$ be an application credential, i.e., a signature on the messages $ID_U, \mathsf{m}_1, \ldots, \mathsf{m}_{L-1}$.

We note that not considering how the application-credential issuer decides whether or not the user is eligible to the credential, is a bit too simplistic. We therefore present in §4.4.1later an extension to our systems a protocol that allows the user to prove possession of other application credentials that the application-credential issuer requires as a prerequisite to issue the requested credential. As we shall see, in this case it is important that all the issued credential and the one that the user proves possession of all encode the same identifier ID_U.

4.2.4 Anonymously Using a Secured Credential

We now describe the procedure UseAppCred.

Let $(\mathsf{A}, \mathsf{e}, \mathsf{v})$ be an application credential, i.e., a signature on the messages $ID_U, \mathsf{m}_1, \ldots, \mathsf{m}_{\mathsf{L}-1}$, that the user wants to prove possession of to a service or resource provider (called verifier in the following) using a device she registered, i.e., for which she got the device credential (A, e) (for which that device's TPM has keeps secret the corresponding m_0, m_1, and v values).

Let $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ be the keys of a commitment scheme described in §3.1 such that the verifier is assured that the user and her platforms are not privy to the factorization of $\mathfrak{n}$.

For simplicity we assume that the verifier learns all the attributes (messages) $\mathsf{m}_1, \ldots \mathsf{m}_{\mathsf{L}-1}$ contained the application credential. However, along the lines of the protocol to prove possession of a CL-signature presented in §3.4, it is easy to modify the below protocol as to hide (some of) the message or to provide the verifier only commitments to (some of) them.

1. The platform computes a commitment $\mathfrak{C}$ to ID_U: it selects a random $\mathfrak{r} \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $\mathfrak{C} := \mathfrak{g}^{ID_U}\mathfrak{h}^{\mathfrak{r}} \bmod \mathfrak{n}$, and sends $\mathfrak{C}$ to the verifier.

2. Using the protocol presented in §3.4 to prove possession of a CL-signature, the platform proves possession of an application credential whose 0-th message is committed to in $\mathfrak{C}$. (The other attributes (i.e., messages) $\mathsf{m}_1, \ldots \mathsf{m}_{\mathsf{L}-1}$ contained in the application credential can be handled as revealed, committed-to, or hidden messages, depending on the application.)

3. To prove that it has also obtained a device-credential, the platform proceeds as follows:

   (a) The platform chooses $r \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $A' := AS^r$, and sends A' to the application-credential issuer.

   (b) The platform as prover (with the help of its TPM) executes the protocol

$$SPK\{(e, \tilde{v}, m_0, m_1, ID_U, \mathfrak{r}) : Z \equiv \pm A'^e S^{\tilde{v}} R_0^{m_0} R_1^{m_1} R_2^{ID_U} \pmod{n} \ \wedge\ \mathfrak{C} \equiv \pm\mathfrak{g}^{ID_U}\mathfrak{h}^{\mathfrak{r}} \bmod \mathfrak{n} \wedge m_0, m_1, ID_U \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_{\mathcal{H}}+2} \ \wedge\ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_{\mathcal{H}}+1}\}$$

   with the application-credential issuer as verifier. To be able to prove the first term, the platform needs to transform the corresponding prover's messages obtained from the TPM via the DAA-protocol in the way described in §4.1 and, in detail, in Appendix A.2.

4.3 Security Analysis

Let us argue that our scheme statisfies unforgeability and anonymity as defined in §2.

Basically, all the security properties of our scheme follow from the security of the Camenisch-Lysyanskaya signature scheme, of the commitment scheme used, and of the direct anonymous attestation protocols and the soundness of the zero-knowledge proof protocols applied. The cryptographic assumptions underlying these schemes are the strong RSA assumption and the decisional Diffie-Hellman assumption modulo a composite. Finally, our scheme further relies on the tamper-resistance of the TPM. However, as is the case with DAA, we do not store system secrets in the TPM but only secrets generated by each TPM (for its user), we argue that this is a resonable assumption as the cost of breaking open a TPM does not justify the gain (cf. [5, 32]).

More precisely, we have the following:

Unforgeability. Assume there is an adversary who breaks the unforgeability property. We argue that such an adversary can either break open a TPM, break the Camenisch-Lysyanskaya signature scheme, or commitment scheme used, or soundness of the zero-knowledge proof protocols. Running the UseAppCred protocol successfully with a verifier means that (using rewinding) one can extract values that form an application-credential and a device credential such that the device credential has a includes as message m_2 the same message that the application credential includes as message m_0, say ID_A. If this is not the case, then either the adversary can open the commitment $\mathcal{C}$ in two different ways (which would violate the discrete logarithm assumption modulo a composite and thus the Diffie-Hellman assumption modulo a composite or that the SPK proofs in Step 2 or 3 are not sound (which would mean that the adversary could break the strong RSA assumption, cf. [21, 9]). Now, from the unforgeability property of the CL signature scheme, it follows that the adversary did engage with the application-credential issuer and the device-credential issuer in the protocols to obtain the respective credentials. Let us consider these protocols and argue that in fact the adversary must have engaged in these protocols with ID_A. Because of the security of the protocol to obtain a signature on a committed message (which follows also from strong RSA assumption and the decisional Diffie-Hellman assumption), the adversary must indeed have run these two protocols with ID_A as hidden message to be signed. Thus it remains to argue that the adversary could not use the device-credential without involving the device, i.e., it's TPM. Now from the properties of the direct anonymous attestation protocols it follows that 1) the device credential is indeed a signature on secrets held by the TPM and 2) these key cannot be extracted from the TPM without breaking it open. Moreover, as proving possession of a signature by the device-credential issuer requires knowledge of the TPM's secret, this proof cannot be done without the involvement of the TPM. This property has been proven for DAA [5] and it not hard to see that it carries over to our scheme (in the end in both scheme knowledge of a signature is proved where some of the messages are held secret inside the TPM).

Thus no such adversary can exist under the strong RSA assumption, the decisional Diffie-Hellman assumption modulo a composite, and the tamper resistance assumption of the TPM.

Anonymity. Due to the zero-knowledgeness of the protocols to proof possession of signature/credentials [9, 5], the service provider does not learn anything about the user except that she has a device-credential and an application credential that encode the same identifier. Also, the application-credential issuer does not learn anything about the user except that he issued her a credential that contains some hidden string as the 0-th message. Thus even if the verifier, application-credential, and the service provider collude, they cannot link different transactions by the same user except the transactions to register devices.

4.4 Extensions

This section describes how our scheme can be extended to secure non-anonymous credentials and to secure a whole credential system.

4.4.1 Extension to a Credential System with Secured Credentials

In section §4.2.3 we have assumed that a user gets an application credential "just like this," i.e., without providing the issuer any information. This is of course not realistic. Rather, the user typically needs to provide (or prove possession of) some other credentials, e.g., to prove that she is of age or to provide some anonymous ecash (which is just a credential that can be used only once). In short, we need to consider a full-fledged credential system. While it should be clear that we can of course tie all such these credential to a TPM, we need actually to make sure that the credential that is issued will be tied to devices of the same user who possesses the credentials required to obtain that credential, cf. [8].

We now describe how this can be achieved. A prerequisite to the protocol is that the user uses one of her devices for which she has obtained a device-credential, say (A, e), and the required application credentials. This protocol

itself can be seen as a combination of the protocol to use an application credential presented in §4.2.4 and the one to obtain a CL-signature on a committed message presented in §3.4.1.

For simplicity we assume that the application credential that will be issued to the user contains the attributes $\mathsf{m}_1, \ldots \mathsf{m}_{\mathsf{L}-1}$ and that they are all known to the application-credential issuer. However, the protocol can be modified such that some of these messages are not know at all to the issuer or only by commitments (cf. protocol to obtain a signature in §3.4).

1. The platform chooses a random $\mathsf{v}' \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $\mathsf{U} := \mathsf{S}^{\mathsf{v}'}\mathsf{R}_0^{ID_U} \bmod \mathsf{n}$ and sends U to application-credential issuer. (This is the first step of the protocol to obtain a signature on a committed message §3.4).

2. The platform shows possession of the device-credential and prove that U commits to the identity encoded in the m_2 of the device credential:

   (a) The platform chooses $r \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $A' := AS^r$, and sends A' to the application-credential issuer.

   (b) The platform as prover executes the protocol

   $$\begin{aligned} \mathit{SPK}\{(e, \tilde{v}, m_0, m_1, \mathsf{v}', ID_U) : \\ Z \equiv \pm A'^e S^{\tilde{v}} R_0^{m_0} R_1^{m_1} R_2^{ID_U} \pmod{n} \ \wedge\ \mathsf{U} \equiv \pm \mathsf{S}^{\mathsf{v}'}\mathsf{R}_0^{ID_U} \bmod \mathsf{n} \wedge \\ m_0, m_1, ID_U \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_{\mathcal{H}}+2} \ \wedge\ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_{\mathcal{H}}+1}\} \end{aligned}$$

   with the application-credential issuer as verifier. To be able to prove the first term, the platform needs to transform the corresponding proof-messages obtained from the TPM in the way described in §4.1 and, in detail, in Appendix A.2.

   Here (as well as in the next step below) we have used R_0 and S instead of $\mathfrak{g}$ and $\mathfrak{h}$ as bases for the commitment scheme. This works because the application-credential issuer (1) know that only he is privy to the factorization of n and (2) he has as part of the setup assure the users that $\mathsf{R}_0 \in \langle \mathsf{S} \rangle$.

3. Using the protocol presented in §3.4 to prove possession of a CL-signature, the platform proves possession of an application credential whose 0-th message is committed to in U. (The other attributes (i.e., messages) contained in the application credential can be handled as revealed, committed-to, or hidden messages, depending on the application.)

4. The application-credential issuer chooses $\hat{\mathsf{v}} \in_R \{0,1\}^{\ell_v-1}$ and a prime $\mathsf{e} \in_R [2^{\ell_e-1}, 2^{\ell_e-1} + 2^{\ell'_e-1}]$, computes $\mathsf{v}'' := \hat{\mathsf{v}} + 2^{\ell_v-1}$ and

   $$\mathsf{A} := \Big(\frac{\mathsf{Z}}{\mathsf{U}\mathsf{S}^{\mathsf{v}''} \prod_{i=1}^{\mathsf{L}-1} \mathsf{R}_i^{\mathsf{m}_i}}\Big)^{1/\mathsf{e}} \bmod \mathsf{n} \ ,$$

   and sends $(\mathsf{A}, \mathsf{e}, \mathsf{v}'')$ to the platform.

5. To convince the platform that A was correctly computed, the application-credential issuer as prover runs the protocol

   $$\mathit{SPK}\{(\mathsf{d}) : \quad \mathsf{A} \equiv \pm\Big(\frac{\mathsf{Z}}{\mathsf{U}\mathsf{S}^{\mathsf{v}''} \prod_{i=1}^{\mathsf{L}-1} \mathsf{R}_i^{\mathsf{m}_i}}\Big)^{\mathsf{d}} \pmod{\mathsf{n}}\}$$

   with the platform as verifier.

6. The platform checks whether e is a prime and lies in $[2^{\ell_e-1}, 2^{\ell_e-1} + 2^{\ell'_e-1}]$. It stores $(\mathsf{A}, \mathsf{e}, \mathsf{v} := \mathsf{v}' + \mathsf{v}'')$ as signature on the messages $ID_U, \mathsf{m}_1, \ldots \mathsf{m}_{\mathsf{L}-1}$.

The above protocol offers perfect anonymity. That is the application-credential issuer does not learn anything about the user except that she has registered a device. In a real application, however, the user probably needs to provide the application-credential issuer some further information to be eligible to obtain an application credential. Of course, this further information could consist of anonymous cash, in which case the user can indeed obtain the application-credential anonymously.

4.4.2 Securing Non-Anonymous Credentials

Our scheme can also be used to secure any non-anonymous credential or certificate as long as they support attributes. The idea is to simply include the user's identifier ID_U as an attribute in the non-anonymous credential (as with any other attribute, the semantics of this attribute must be known to all involved parties which can be done, e.g., via the certificate of the signer's public key).

To use such a credential with any registered device, the user reveals ID_U together with the non-anonymous credential, and then shows that ID_U is actually included in the device-credential. To do the latter, the user's device's platform extends the TPM's prover's messages from the DAA protocol to the following one (cf. §A.2 as well as the protocol to show knowledge of a signature that is tied to a TPM presented above):

$$PK\{(e,\tilde{v},m_0,m_1): \quad \frac{Z}{R_2^{ID_U}} \equiv \pm A'^{e} S^{\tilde{v}} R_i^{m_0} R_i^{m_1} \pmod{n} \wedge$$
$$m_0, m_1 \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_{\mathcal{H}}+2} \ \wedge \ (e-2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_{\mathcal{H}}+1}\} \ ,$$

Apart from verifying this proof, the verifier needs of course to check as well that ID_U is indeed contained in the non-anonymous credential (in the right place).

4.5 Experimental Results

A prototype of our scheme has been implemented in the in the JAVA programming language. That is, all the parties were implemented in software, in particular also the TPM as such chip that implement the DAA protocol as currently not yet available. For all group operations, the standard JAVA BigInteger class was used.

On a IBM Thinkpad T41 with a 1.7 GHz Intel Mobile Pentium M processor and 1 GByte of RAM, running Linux and IBM Java Runtime Environment 1.4.2, we have made the following measurements. The protocol to register a device required about 2.5 seconds of running time in total (excluding communication time), about 25% of the time was used by the TPM, about 25% was used by the platform, and about 50% was used by the issuer. The protocol to obtain an application credential required slightly less running time, the time being used on the issuer and the platform being about equal. The protocol to use an application credential required about 6-7-seconds of running time in total (excluding communication time), about 10% of the time was used by the TPM, about 45% was used by the platform, and about 45% was used by the verifier.

However, as mentioned we used the standard JAVA BigInteger class only and therefore these running times can be significantly reduced if good (multi-base) exponentiation algorithms are used; we expect all these operations to be below one second. Also, all the protocols and algorithms were implemented as single threaded programs. However, in a real application the parties can make (some of) their computations in parallel.

References

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology — CRYPTO 2000*, vol. 1880 of *LNCS*, pp. 255–270. Springer Verlag, 2000.

[2] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM CCS*, pp. 244–250, 1993.

[3] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology — EUROCRYPT 2000*, vol. 1807 of *LNCS*, pp. 431–444. Springer Verlag, 2000.

[4] S. Brands. Rapid demonstration of linear relations connected by boolean operators. In *Advances in Cryptology — EUROCRYPT '97*, vol. 1233 of *LNCS*, pp. 318–333. Springer Verlag, 1997.

[5] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *11th ACM Conference on Computer and Communicatons Security*. Association for Computing Machinery, Oct. 2004.

[6] J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In *Advances in Cryptology — ASIACRYPT 2000*, vol. 1976 of *LNCS*, pp. 331–345. Springer Verlag, 2000.

[7] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *Security in Communication Networks, Forth International Conference, SCN 2004*, LNCS. Springer Verlag, 2004.

[8] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *Advances in Cryptology — EUROCRYPT 2001*, vol. 2045 of *LNCS*, pp. 93–118. Springer Verlag, 2001.

[9] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks, Third International Conference, SCN 2002*, vol. 2576 of *LNCS*, pp. 268–289. Springer Verlag, 2003.

[10] J. Camenisch and M. Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *Advances in Cryptology — EUROCRYPT '99*, vol. 1592 of *LNCS*, pp. 107–122. Springer Verlag, 1999.

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

[12] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology — CRYPTO 2003*, vol. 2729 of *LNCS*, pp. 126–144, 2003.

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

[14] J. L. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zürich, 1998. Diss. ETH No. 12520, Hartung Gorre Verlag, Konstanz.

[15] A. Chan, Y. Frankel, and Y. Tsiounis. Easy come – easy go divisible cash. In *Advances in Cryptology — EUROCRYPT '98*, vol. 1403 of *LNCS*, pp. 561–575. Springer Verlag, 1998.

[16] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, Oct. 1985.

[17] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology — CRYPTO '92*, vol. 740 of *LNCS*, pp. 89–105. Springer-Verlag, 1993.

[18] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology — EUROCRYPT '91*, vol. 547 of *LNCS*, pp. 257–265. Springer-Verlag, 1991.

[19] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology — CRYPTO '94*, vol. 839 of *LNCS*, pp. 174–187. Springer Verlag, 1994.

[20] I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. `http://eprint.iacr.org/2001/064`, 2001.

[21] I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology — ASIACRYPT 2002*, vol. 2501 of *LNCS*. Springer, 2002.

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

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

[24] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In *Advances in Cryptology — EUROCRYPT 2003*, vol. 2656 of *LNCS*, pp. 524–543. Springer Verlag, 2003.

[25] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988.

[26] J. Kilian and E. Petrank. Identity escrow. In *Advances in Cryptology — CRYPTO '98*, vol. 1642 of *LNCS*, pp. 169–185, Berlin, 1998. Springer Verlag.

[27] P. R. Mihir Bellare, David Pointcheval. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology — EUROCRYPT 2000*, vol. 1087 of *LNCS*, pp. 139–155. Springer, 2000.

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

[29] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[30] C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[31] Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification, version 1.1a. Republished as Trusted Computing Group (TCG) main specifcation, Version 1.1b, Available at `www.trustedcomputinggroup.org`, 2001.

[32] Trusted Computing Group. TCG TPM specification 1.2. Available at `www.trustedcomputinggroup.org`, 2003.

[33] Trusted Computing Group website. `www.trustedcomputinggroup.org`.

[34] S. P. Victor Boyko, Philip D. MacKenzie. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology — EUROCRYPT 2000*, vol. 1087 of *LNCS*, pp. 156–171. Springer, 2000.

A Details of How the Platform can Extend DAA

A.1 Obtaining a Signature that is tied to a TPM

This section provides the details of the protocol that allows the TPM and platform to obtain an extended DAA-credential as described on a high level in §4.1.1.

The input to the protocol are the public key of the DAA-issuer. The platform's input further consists of the messages $m_2, \ldots, m_{L-1} \in \pm\{0,1\}^{\ell_m}$.

1. The TPM chooses m_0 and m_1 (see [5] for details on this) and $v' \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes $U := R_0^{m_0} R_1^{m_1} S^{v'} \bmod n$ and $N_I := \zeta_I^{m_0+m_1 2^{\ell_m}} \bmod \Gamma$ and sends U and N_I to the platform.
2. The platform computes sends U and N_I to the $U' := S^{\hat{v}} \prod_{i=2}^{L'-1} R_i^{m_i} \bmod n$ issuer, where $\hat{v} \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$.
3. Now, the TPM and the platform generate the verifier parts of the proof protocol

$$\begin{aligned} \textit{SPK}\{(m_0,\ldots,m_{L'-1},v',\hat{v}) : U \equiv \pm S^{v'} R_0^{m_0} R_1^{m_1} \pmod n \ \wedge\ N_I := \zeta_I^{m_0+m_1 2^{\ell_m}} \bmod \Gamma \ \wedge \\ U' \equiv \pm S^{\hat{v}} \prod_{i=2}^{L'-1} R_i^{m_i} \pmod n \ \wedge \\ m_0,\ldots,m_{L'-1} \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_\mathcal{H}+2} \ \wedge\ v' \in \{0,1\}^{\ell_n+\ell_\varnothing+\ell_\mathcal{H}+2}\} \end{aligned}$$

as follows.

   (a) The TPM picks random integers $r_{m_0}, r_{m_1} \in_R \{0,1\}^{\ell_m+\ell_\varnothing+\ell_\mathcal{H}}$, $r_{v'} \in_R \{0,1\}^{\ell_n+2\ell_\varnothing+\ell_\mathcal{H}}$, and computes $\tilde{U} := R_0^{r_{m_0}} R_1^{r_{m_1}} S^{r_{v'}} \bmod n$ and $\tilde{N}_I := \zeta_I^{r_{m_0}+r_{m_1}2^{\ell_m}} \bmod \Gamma$, and sends $\tilde{U}'$ and $\tilde{N}_I$ to the platform.

   (b) The platform chooses $r_{m_2}, \ldots, r_{L'-1} \in_R \{0,1\}^{\ell_m+\ell_\varnothing+\ell_\mathcal{H}}$ and $r_{\hat{v}} \in_R \{0,1\}^{\ell_m+2\ell_\varnothing+\ell_\mathcal{H}}$ and computes $\tilde{U}' := S^{r_{\hat{v}}} \prod_{i=2}^{L'+1} R_i^{r_{m_i}} \bmod n$.

   (c) The issuer chooses a random string $n_i \in \{0,1\}^{\ell_\mathcal{H}}$ and sends n_i to the platform.

   (d) The platform computes $c_h := H(n \| R_0 \| \ldots \| R_{L'-1} \| S \| U \| U' \| N_I \| \tilde{U} \| \tilde{U}' \| \tilde{N}_I \| n_i)$ and sends c_h to the TPM

   (e) The TPM chooses a random $n_t \in \{0,1\}^{\ell_\varnothing}$ and computes

$$c := H(c_h \| n_t) \in [0, 2^{\ell_\mathcal{H}} - 1] \ .$$

   (f) The TPM computes

$$s_{m_0} := r_{m_0} + c \cdot m_0 \ , \qquad s_{m_1} := r_{m_1} + c \cdot m_1 \ , \qquad s_{v'} := r_{v'} + c \cdot v' \ . \tag{1}$$

   (g) TPM sends $(c, n_t, s_{m_0}, s_{m_1}, s_{v'})$ to the platform.

   (h) The platform computes

$$s_{a_i} := r_{a_i} + c \cdot a_i \qquad \text{for} \qquad i = 1, \ldots, \ell_h \ , \qquad s_{\tilde{v}'} := r_{\tilde{v}'} + c \cdot \tilde{v}' \ . \tag{2}$$

   and sends $(c, n_t, s_{m_0}, s_{m_1}, s_{v'}, s_{a_1}, \ldots, s_{a_{\ell_h}})$ to the issuer.

(i) Issuer verifies the proof as follows. It computes

$$\hat{U} := U^{-c} R_0^{s_{m_0}} R_1^{s_{m_1}} S^{s_{v'}} \bmod n \ , \tag{3}$$

$$\hat{U}' := U'^{-c} S^{s_{\tilde{v}}} \prod_{i=2}^{L'-1} R_i^{s_{m_i}} \bmod n \ , \tag{4}$$

$$\hat{N}_I := N_I^{-c} \zeta_I^{s_{m_0} + 2^{\ell_f} s_{m_1}} \bmod \Gamma \tag{5}$$

and checks

$$c \stackrel{?}{=} H(H(n \| R_0 \| \ldots \| R_{L'-1} \| S \| U \| U' \| N_I \| \hat{U} \| \hat{U}' \| \hat{N}_I \| n_i) \| n_t) \in [0, 2^{\ell_{\mathcal{H}}} - 1] \ , \tag{6}$$

$$s_{m_0}, \ldots, s_{m_{L'-1}} \stackrel{?}{\in} \{0,1\}^{\ell_m + \ell_\varnothing + \ell_{\mathcal{H}} + 1} \ , \quad \text{and} \quad s_{v'}, s_{\tilde{v}'} \stackrel{?}{\in} \{0,1\}^{\ell_n + 2\ell_\varnothing + \ell_{\mathcal{H}} + 1} \ . \tag{7}$$

4. The issuer chooses $\hat{v} \in_R \{0,1\}^{\ell_v - 1}$ and a prime $e \in_R [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$ and computes

$$v'' := \hat{v} + 2^{\ell_v - 1} \quad \text{and} \quad A := \Big(\frac{Z}{UU' S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^{1/e} \bmod n \ .$$

5. To convince the platform that A was correctly computed, the issuer as prover runs the protocol

$$\textit{SPK}\{(d) : \quad A \equiv \pm \Big(\frac{Z}{UU' S^{v''} \prod_{i=\ell_h+1}^{\ell_h+\ell_i} R_{i+1}^{a_i}}\Big)^d \pmod n\}(n_h)$$

with the platform:

(a) The platform chooses a random integer $n_h \in \{0,1\}^{\ell_\varnothing}$ and sends n_h to the issuer.

(b) The issuer randomly chooses $r_e \in_R [0, p'q']$ and computes

$$\tilde{A} := \Big(\frac{Z}{UU' S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^{r_e} \bmod n \ , \tag{8}$$

$$c' := H(n \| Z \| S \| U \| R_{L'}, \| \ldots R_{L-1} \| v'' \| A \| \tilde{A} \| n_h) \ , \quad \text{and} \tag{9}$$

$$s_e := r_e - c'/e \bmod p'q' \tag{10}$$

and sends $a_{\ell_h+1}, \ldots, a_{\ell_h+\ell_i}$, c', s_e, and (A, e, v'') to the platform.

(c) The platform verifies whether e is a prime and lies in $[2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$, computes

$$\hat{A} := A^{c'} \Big(\frac{Z}{US^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^{s_e} \bmod n \ ,$$

and checks whether

$$c' \stackrel{?}{=} H(n \| Z \| S \| U \| R_{L'}, \| \ldots R_{L-1} \| v'' \| A \| \hat{A} \| n_h) \quad \text{and} \quad Z \stackrel{?}{\equiv} A^e U S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i} \pmod n \tag{11}$$

hold.

6. The platform forwards v'' to the TPM.

7. The TPM receives v'', sets $v := v'' + v'$, and stores (m_0, m_1, v).

8. The platform stores A, e, $\hat{v}$ and $m_2, \ldots, m_L$.

We note that the operations performed by the TPM are exactly those as in the direct anonymous attestation DAA-sign protocol [5] (we have used a somewhat different naming of variables here, however).

A.2 Showing an Extended DAA-Credential

This section provides the details of the protocol that allows the TPM and platform to prove possession an extended DAA-credential as described on a high level in §4.1.2. That is, we provide the protocol that the platform can run with the TPM to obtain the following SPK-signature.

$$\begin{aligned} &\mathit{SPK}\{(e, \tilde{v}, \{m_i | i \in I_c \cup I_h\}) : \\ &\qquad Z \prod_{i \in I_r} R_i^{-m_i} \equiv \pm A'^{e} S^{\tilde{v}} \prod_{i \in I_c \cup I_h} R_i^{m_i} \pmod{n} \ \wedge \ C_i \equiv \mathfrak{g}^{m_i} \mathfrak{h}^{r_i} \pmod{\mathfrak{n}} \ (i \in I_c) \ \wedge \\ &\qquad\qquad m_i \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H} + 2} \ (i \in I_c \cup I_h) \ \wedge \ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_\mathcal{H} + 1} \} \ , \end{aligned}$$

where, as described in §3.4, the value A' is derived from A, the values C_i are commitments to messages, and I_c and I_h are set of indices.

Besides the various public keys, the credential, and the required secrets of the TPM and platform, the input to the protocol consists also of a base $\zeta \in \mathbb{Z}_\Gamma^*$ and a nonce n_v generated by the verifier. For details on how the base ζ is generated we refer to [5].

1. (a) For $i \in I_c$, the platform chooses $r_i \in \{0,1\}^{\ell_n + \ell_\varnothing}$, computes $C_i := \mathfrak{g}^{m_i} \mathfrak{h}^{r_i} \bmod \mathfrak{n}$, and sends C_i to $\mathcal{V}$ and $\{m_i | i \in I_r\}$. The platform chooses $r \in_R \{0,1\}^{\ell_n + \ell_\varnothing}$, computes $A' := AS^r$, and sends A' to $\mathcal{V}$.

   (b) The TPM computes $N_V := \zeta^{m_0 + m_1 2^{\ell_m}} \bmod \Gamma$ and sends N_V to the platform.

2. (a) i. The TPM picks random integers $r_v \in_R \{0,1\}^{\ell_v + \ell_\varnothing + \ell_\mathcal{H}}$ and $r_{m_0}, r_{m_1} \in_R \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H}}$, and computes

$$\tilde{U} := R_0^{r_{m_0}} R_1^{r_{m_1}} S^{r_v} \bmod n \ , \qquad \tilde{r}_m := r_{m_0} + r_{m_1} 2^{\ell_m} \bmod \rho \ , \qquad \tilde{N}_V := \zeta^{\tilde{r}_m} \bmod \Gamma \ .$$

   The TPM sends $\tilde{U}$ and $\tilde{N}_V$ to the platform.

   ii. The platform picks random integers

$$\begin{aligned} r_e &\in_R \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_\mathcal{H}} \ , & r_r &\in_R \{0,1\}^{\ell_e + \ell_n + 2\ell_\varnothing + \ell_\mathcal{H} + 1} \ , \\ r_{m_i} &\in_R \{0,1\}^{\ell_m + \ell_\varnothing + \ell_\mathcal{H}} \ (i \in I_c \cup I_h) \ , & r_{r_i} &\in_R \{0,1\}^{\ell_n + 2\ell_\varnothing + \ell_\mathcal{H}} \ (i \in I_c) \ , \end{aligned}$$

   and computes

$$\tilde{A} := \tilde{U} A'^{r_e} S^{r_r} \prod_{i \in I_c \cup I_h} R_i^{r_{m_i}} \bmod n \qquad \tilde{C}_i := \mathfrak{g}^{r_{m_i}} \mathfrak{h}^{r_{r_i}} \bmod \mathfrak{n} \quad (i \in I_c)$$

   (b) i. The platform computes

$$c_h := H((n \| g \| g' \| h \| R_0 \| R_1 \| S \| Z \| \gamma \| \Gamma \| \rho) \| \zeta \| A' \| N_V \| \{C_i\}_{i \in I_c} \| \tilde{A} \| \tilde{N}_V) \| \{\tilde{C}_i\}_{i \in I_c} \| n_v) \ .$$

   and sends c_h to the TPM.

   ii. The TPM chooses a random $n_t \in \{0,1\}^{\ell_\varnothing}$, computes $c := H(H(c_h \| n_t) \| b \| m)$, and sends c, n_t to the platform.

   (c) i. The TPM computes (over the integers)

$$s_v := r_v + c \cdot v \ , \qquad s_{m_0} := r_{m_0} + c \cdot m_0 \ , \quad \text{and} \qquad s_{m_1} := r_{m_1} + c \cdot m_1$$

   and sends (s_v, s_{m_0}, s_{m_1}) to the platform.

ii. The platform computes (over the integers)

$$s_e := r_e + c \cdot (e - 2^{\ell_e - 1}) \ , \qquad s_{\tilde{v}} := s_v + r_r + c \cdot (r \cdot e) \ ,$$
$$s_{m_i} := r_{m_i} + c \cdot (e - 2^{\ell_e - 1}) \quad (i \in I_c \cup I_h) \ , \qquad s_{r_i} := r_{r_i} + c \cdot (r \cdot e) \quad (i \in I_c) \ .$$

3. The platform outputs the signature

$$\sigma := (\zeta, A', N_V, c, n_t, (s_{\tilde{v}}, s_{m_0}, s_{m_1}, s_e, \{s_{m_i}\}_{i \in I_c \cup I_h}, \{s_{r_i}\}_{i \in I_c}) \ .$$

As for the protocol to obtain a signature, we note that the operations performed by the TPM are exactly those as in the direct anonymous attestation DAA-sign protocol [5] (we have used a somewhat different naming of variables here, however).

Such an signature

$$\sigma := (\zeta, A', N_V, c, n_t, (s_{\tilde{v}}, s_{m_0}, s_{m_1}, s_e, \{s_{m_i}\}_{i \in I_c \cup I_h}, \{s_{r_i}\}_{i \in I_c}) \ .$$

is verified as follows (cf. [5]).

1. Compute

$$\hat{A} := (Z \prod_{i \in I_r} R_i^{-m_i})^{-c} A'^{s_e} S^{s_{\tilde{v}}} \prod_{i \in \{0,1\} \cup I_c \cup I_h} R_i^{r_{m_i}} \bmod n$$
$$\hat{C}_i := C_i^{-c} \mathfrak{g}^{s_{m_i}} \mathfrak{h}^{s_{r_i}} \bmod \mathfrak{n} \quad (i \in I_c)$$
$$\hat{N}_V := N_V^{-c} \zeta^{s_{m_0} + s_{m_1} 2^{\ell_m}} \bmod \Gamma \ .$$

2. Verify that

$$c \stackrel{?}{=} H(H(H((n \| g \| g' \| h \| R_0 \| R_1 \| S \| Z \| \gamma \| \Gamma \| \rho) \| \zeta \| A' \| N_V \| \{C_i\}_{i \in I_c} \| \hat{A} \| \hat{N}_V) \| \{\hat{C}_i\}_{i \in I_c} \| n_v) \| n_t) \| b \| m) \ ,$$
$$N_V, \zeta \stackrel{?}{\in} \langle \gamma \rangle \ , \quad s_{m_i} \stackrel{?}{\in} \{0,1\}^{\ell_m + \ell_\varnothing + \ell_{\mathcal{H}} + 1} \quad (i \in \{0,1\} \cup I_c \cup I_h) \ , \quad \text{and} \quad s_e \stackrel{?}{\in} \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_{\mathcal{H}} + 1} \ .$$

For the checks involving N_V and ζ to detect whether a TPM might be a rogue we refer to [5].