Direct Anonymous Attestation[*]

Ernie Brickell
Intel Corporation
ernie.brickell@intel.com

Jan Camenisch
IBM Research
jca@zurich.ibm.com

Liqun Chen
HP Laboratories
liqun.chen@hp.com

ABSTRACT

This paper describes the direct anonymous attestation scheme (DAA). This scheme was adopted by the Trusted Computing Group (TCG) as the method for remote authentication of a hardware module, called Trusted Platform Module (TPM), while preserving the privacy of the user of the platform that contains the module. DAA can be seen as a group signature without the feature that a signature can be opened, i.e., the anonymity is not revocable. Moreover, DAA allows for pseudonyms, i.e., for each signature a user (in agreement with the recipient of the signature) can decide whether or not the signature should be linkable to another signature. DAA furthermore allows for detection of "known" keys: if the DAA secret keys are extracted from a TPM and published, a verifier can detect that a signature was produced using these secret keys. The scheme is provably secure in the random oracle model under the strong RSA and the decisional Diffie-Hellman assumption.

Categories and Subject Descriptors

x.x.x [**Data**]: Data Encryption—*Standards*

General Terms

Standardization,Algorithms,Security

Keywords

Privacy, Anonymous Credential Systems, Cryptographic Protocols, Trusted Computing, Integrity Based Computing.

1. INTRODUCTION

Consider a trusted hardware module, called the trusted platform module (TPM) in the following, that is integrated into a platform such as a laptop or a mobile phone. Assume that the user of such a platform communicates with a verifier who wants to be assured that the user indeed uses a platform containing such a trusted hardware module, i.e., the verifier wants the TPM to authenticate itself. However, the user wants her privacy protected and therefore requires that the verifier only learns that she uses a TPM but not which particular one – otherwise all her transactions would become linkable to each other. This problem arose in the context of the Trusted Computing Group (TCG). TCG is an industry standardization body that aims to develop and promote an open industry standard for trusted computing hardware and software building blocks to enable more secure data storage, online business practices, and online commerce transactions while protecting privacy and individual rights (cf. [37]). TCG is the successor organization of the Trusted Computing Platform Alliance (TCPA).

In principle, the problem just described could be solved using any standard public key authentication scheme (or signature scheme): One would generate a secret/public key pair, and then embed the secret key into each TPM. The verifier and the TPM would then run the authentication protocol. Because all TPMs use the same key, they are indistinguishable. However, this approach would never work in practice: as soon as one hardware module (TPM) gets compromised and the secret key extracted and published, verifiers can no longer distinguish between real TPMs and fake ones. Therefore, detection of rogue TPMs needs to be a further requirement.

The solution first developed by TCG uses a trusted third party, the so-called privacy certification authority (Privacy CA), and works as follows [35]. Each TPM generates an RSA key pair called an Endorsement Key (EK). The Privacy CA is assumed to know the Endorsement Keys of all (valid) TPMs. Now, when a TPM needs to authenticate itself to a verifier, it generates a second RSA key pair called an Attestation Identity Key (AIK), sends the AIK public key to the Privacy CA, and authenticates this public key w.r.t. the EK. The Privacy CA will check whether it finds the EK in its list and, if so, issues a certificate on the TPM's AIK. The TPM can then forward this certificate to the verifier and authenticate itself w.r.t. this AIK. In this solution, there are two possibilities to detect a rogue TPM: 1) If the EK secret key was extracted from a TPM, distributed, and then detected and announced as a rogue secret key, the Privacy CA can compute the corresponding public key and remove it from its list of valid Endorsement Keys. 2) If the Privacy CA gets many requests that are authorized using the same Endorsement Key, it might want to reject these requests. The exact threshold on requests that are allowed before a

[*]A full version of this paper with a full security proof is available at http://eprint.iacr.org/2004/205/

CCS'04, October 25-29, 2004, Washington, DC, USA.
Copyright 2004 ACM 1-58113-961-6/04/0010 ...$5.00.

TPM is tagged rogue depends of course on the actual environment and applications, and will in practise probably be determined by some risk-management policy.

This solutions has the obvious drawback that the Privacy CA needs to be involved in every transaction and thus highly available on the one hand but still as secure as an ordinary certification authority that normally operates off-line on the other hand. Moreover, if the Privacy CA and the verifier collude, or the Privacy CA's transaction records are revealed to the verifier by some other means, the verifier will still be able to uniquely identify a TPM. Although the latter problem could be solved by using blind signatures [17] instead of ordinary signatures in the private CA's certification, the first problem, the private CA's availability, would persist in such a solution.

In this paper, we describe a better solution that was adopted by TCG in the new specification of the TPM [36]. It draws on techniques that have been developed for group signatures [22, 14, 1], identity escrow [29], and credential systems [18, 7]. In fact, our scheme 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 our case). More precisely, we also employ a suitable signature scheme to issue certificates on a membership public key generated by a TPM. Then, to authenticate as a group member, or valid TPM, a TPM proves that it possesses a certificate on a public key for which it also knows the secret key. To allow a verifier to detect rogue TPMs, the TPM is further required to reveal and prove correct of a value $N_V = \zeta^f$, where f is its secret key and ζ is a generator of an algebraic group where computing discrete logarithms is infeasible. As in the Privacy-CA solution, there are two possibilities for the verifier to detect a rogue TPM: 1) By comparing N_V with $\zeta^{\tilde{f}}$ for all $\tilde{f}$'s that are known to stem from rogue TPMs. 2) By detecting whether he has seen the same N_V too many times. Of course, the second method only works if the same ζ is used many times. However, ζ should not be a fixed system parameter as otherwise the user gains almost no privacy. Instead, ζ should either be randomly chosen by the TPM each time when it authenticates itself or every verifier should use a different ζ and change it with some frequency. Whether a verifier allows a TPM to choose a random base ζ and, if not, how often a verifier changes its ζ is again a question of risk management and policies and is outside the scope of this paper. However, we assume in the following that in case ζ is not random, it is derived from the verifier's name, e.g., using an appropriate hash function.

Because the TPM is a small chip with limited resources, a requirement for direct anonymous attestation was that the operations carried out on the TPM be minimal and, if possible, be outsourced to (software that is run on) the TPM's host. Of course, security must be maintained, i.e., a (corrupted) host/software should not be able to authenticate without interacting with the TPM. However, privacy/anonymity needs only be guaranteed if the host/software is not corrupted: as the host controls all the communication of the TPM to the outside, a corrupted host/software can always break privacy/anonymity by just adding some identifier to each message sent by the TPM. In fact, our scheme satisfies an even stronger requirement: when the corrupted software is removed, the privacy/anonymity properties are restored.

As our scheme employs the Camenisch-Lysyanskaya signature scheme [8, 31] and the respective discrete logarithms based proofs to prove possession of a certificate, unforgeability of certificates holds under the strong RSA assumption and privacy and anonymity is guaranteed under the decisional Diffie-Hellman assumption. Furthermore, we use the Fiat-Shamir heuristic to turn proofs into signatures and thus our security proofs also assume random oracles.

As already mentioned, our setting shares many properties with the one of group signatures [22, 14, 1], identity escrow [29], and credential systems [18, 7] and we employ many techniques [1, 7, 8] used in these schemes. However, unlike those schemes, the privacy/anonymity properties do not require that the issuer uses so-called safe primes. We achieve this by a special sub-protocol when issuing credentials. This rids us of the necessity that the issuer proves that his RSA modulus is a safe-prime product which makes the setup of those schemes rather inefficient.

Finally, there have been two other methods [5, 3] presented to TCG that address the same problem as we do. The security of Brickell's direct proof method [5] is based on the Bounded Decision Diffie-Hellman assumption and a new assumption called the Interval RSA assumption stating that given e and n, it is hard to find a pair (x, y) such that $x = y^e \pmod{n})$ and x lies in some small specific interval. Besides this non-standard assumption, this direct proof method is less efficient than ours (it uses cut-and-choose proofs). The security of the Boneh, Brickell, Chen, Shacham set signature method [3] is based on the strong RSA assumption and the Bounded Decision Diffie-Hellman assumption, and is based on the group signature scheme by Ateniese et al. [1]. However, neither of these methods [5, 3] are proven to protect against a corrupted TPM when issuing certificates. This may be acceptable in the case that a TPM does not leave the control of a secure manufacturing facility before a certificate is issued. However, to meet a requirement that certificates may be issued to TPMs after they have been released to an insecure environment, it is desirable to use a scheme in which the signature scheme used to issue certificates is proven to be secure against adaptive chosen message attacks, which is a property we prove for our scheme.

2. FORMAL SPECIFICATION OF DIRECT ANONYMOUS ATTESTATION AND SECURITY MODEL

This section provides the formal model of direct anonymous attestation (DAA). As in [7], we use an ideal-system/real-system model to prove security based on security models for multi-party computation [15, 16] and reactive systems [32, 33].

We summarize the ideas underlying these models. In the real system there are a number of players, who run some cryptographic protocols with each other, an adversary $\mathcal{A}$, who controls some of the players, and an environment $\mathcal{E}$ that 1) provides the player $\mathcal{U}_i$ with inputs and 2) arbitrarily interacts with $\mathcal{A}$. The environment provides the inputs to the honest players and receives their outputs and interacts arbitrarily with the adversary. The dishonest players are subsumed into the adversary.

In the ideal system, we have the same players. However, they do not run any cryptographic protocol but send all their inputs to and receive all their outputs from an ideal all-trusted party $\mathcal{T}$. This party computes the output of the players from their inputs, i.e., applies the functionality that the cryptographic protocols are supposed to realize.

A cryptographic protocol is said to implement securely a functionality if for every adversary $\mathcal{A}$ and every environment $\mathcal{E}$ there exists a simulator $\mathcal{S}$ controlling the same parties in the ideal system as $\mathcal{A}$ does in the real system such that the environment can not distinguish whether it is run in the real system and interacts with $\mathcal{A}$ or whether it is run in the ideal system and interacts with the simulator $\mathcal{S}$.

We now specify the functionality of direct anonymous attestation. We distinguish the following kinds of players: the issuer $\mathcal{I}$, a trusted platform module (TPM) $\mathcal{M}_i$ with identity $\texttt{id}_i$, a host $\mathcal{H}_i$ that has TPM $\mathcal{M}_i$ "built in," the rogue detection oracle $\mathcal{O}$ announcing which TPMs are rogue, and verifiers $\mathcal{V}_j$.

In the following specification, a counter value cnt is used to allow a TPM to generate multiple DAA keys from a single secret and a basename bsn is used to provide the property of a possible link, which is controlled by a signer and a verifier, between multiple DAA signatures signed under the same DAA key. Borrowing terminology from group signatures, by "join" we denote the procedure in which a TPM gets issued an anonymous certificate and there "joins" the group of certified or attested TPMs. By "DAA-Sign/Verify" we denote the procedure with which the TPM and its platform can convince a verifier that the TPM is certified. We use the word "sign" as the verifier gets as a result of this procedure a piece of information that he or she can use to convince another entity that he or she was communicating with a certified TPM and also because the procedure can indeed be used to sign messages, in particular, attestation identity keys (AIK) generated by the very same TPM. Finally, the "rogue tagging" operation corresponds to the event that one finds a TPM's DAA keys, e.g., on the Internet, and wants to publish those keys as invalid.

The ideal system all-trusted party $\mathcal{T}$ supports the following operations:

Setup: Each player indicates to $\mathcal{T}$ whether or not it is corrupted. Each TPM $\mathcal{M}_i$ sends its unique identity $\texttt{id}_i$ to $\mathcal{T}$ who forwards it to the respective host $\mathcal{H}_i$.

Join: The host $\mathcal{H}_i$ contacts $\mathcal{T}$ and requests to become a member w.r.t. to a counter value cnt. Thus $\mathcal{T}$ sends the corresponding TPM $\mathcal{M}_i$ the counter value cnt and asks it whether it wants to become a member w.r.t. counter value cnt. Then, $\mathcal{T}$ asks the issuer $\mathcal{I}$ whether the platform with identity id and counter value cnt can become a member. If $\mathcal{M}_i$ was tagged rogue w.r.t. some counter value, $\mathcal{T}$ also tell $\mathcal{I}$ this. If the issuer agrees, $\mathcal{T}$ notifies $\mathcal{H}_i$ that it has become a member.

DAA-Sign/Verify: A host $\mathcal{H}_i$ wants to sign a message m for some verifier $\mathcal{V}_j$ with respect to some counter value cnt and some basename $\texttt{bsn} \in \{0,1\}^* \cup \{\perp\}$ (If a signature is done w.r.t. a basename we have $\texttt{bsn} \in \{0,1\}^*$ and if its done w.r.t. no basename we set $\texttt{bsn} = \perp$). So $\mathcal{H}_i$ sends m, bsn and cnt to $\mathcal{T}$. If $\mathcal{H}_i/\mathcal{M}_i$ are not a member w.r.t. cnt, then $\mathcal{T}$ denies the request. Otherwise, $\mathcal{T}$ forwards m and cnt to the corresponding $\mathcal{M}_i$ and asks it whether it wants to sign. If it does, $\mathcal{T}$ tells $\mathcal{H}_i$ that $\mathcal{M}_i$ agrees and asks it w.r.t. which basename bsn it wants to sign (or whether it wants to abort). If $\mathcal{H}_i$ does not abort, $\mathcal{T}$ proceeds as follows

- If $\mathcal{M}_i$ has been tagged rogue w.r.t. cnt, $\mathcal{T}$ lets $\mathcal{V}_j$ know that a rogue TPM has signed m.
- If $\texttt{bsn} = \perp$ then $\mathcal{T}$ informs $\mathcal{V}_j$ that m has been signed w.r.t. bsn.
- If $\texttt{bsn} \neq \perp$ then $\mathcal{T}$ checks whether $\mathcal{H}_i/\mathcal{M}_i$ have already signed a message w.r.t. bsn and cnt. If this is the case, $\mathcal{T}$ looks up the corresponding pseudonym P in its database; otherwise $\mathcal{T}$ chooses a new random pseudonym $P \in_R \{0,1\}^{\ell_\sigma}$ (the quantity ℓ_σ is a security parameter). Finally, $\mathcal{T}$ informs $\mathcal{V}_j$ that the platform with pseudonym P has signed m.

Rogue Tagging: $\mathcal{O}$ tells $\mathcal{T}$ to tag of the platform with identity id w.r.t. cnt as a rogue. If the TPM with identity id is not corrupted, $\mathcal{T}$ denies the request. Otherwise, $\mathcal{T}$ marks the TPM with identity id as rogue w.r.t. counter value cnt.

Let us discuss our model. Note that the ideal system model captures both unforgeability and anonymity/pseudonymity. More precisely, a signature can only be produced with the involvement of a TPM that is a member and is not tagged rogue. Furthermore, signatures involving the same TPM w.r.t. the same basename are linkable to each other via a pseudonym P, but if they are done w.r.t. different basenames or no basename then they cannot be linked. These two properties hold, regardless of whether or not the corresponding host is corrupted. Anonymity/pseudonymity is only guaranteed if both the host and the TPM are honest, as a dishonest party can always announce its identity and the messages it signed.

The way we have modeled rogue-tagging of a TPM has no effect on old messages but only causes verifiers to reject messages signed by a TPM that is tagged rogue. In the cryptographic protocol, however, an (honest) verifier would in principle be able to identify the messages a rogue TPM signed before it gets tagged as rogue. For simplicity, we do not model this and thus an honest verifier in the real system will not consider this information.

In the model, we assumed that no two verifiers use the same basename bsn. If they do, we consider them to be the same entity. But of course, a verifier can use several different basenames.

We also assume that TPM has a unique identity id with respect to which can identify itself. However, the issuer can always add new hosts/TPMs to the system and hence the number of hosts/TPMs is not fixed. As all these hosts/TPMs have some external identifier, we do not model such additions but just assume that there is always another host/TPM if we need one. Also, in real life, the issuer will decide whether or not a given TPM is allowed to become a member based upon some policy (e.g., the issuer might only allow a specific TPM to join w.r.t., e.g., ten different counter values) and based upon a list of the identities of the valid TPMs.

In our security proofs we make two assumptions about the adversary: First, we assume that the rogue-tagging oracle $\mathcal{O}$ is always controlled by the adversary. The rationale behind this is that the rogue-tagging oracle models the case where a rogue hardware module from a platform is found, its keys are extracted and published on a rogue list. Thus the adversary "controls" this oracle as the adversary can decide when such a rogue platform is found. Note that this assumption strengthens the model. Second, we will not consider corrupted TPMs embedded into honest hosts. The reason is that we assume that at some point all platforms, i.e., hosts and TPM are genuine. Once the platforms get shipped, they might get compromised. As it is easier to compromise a platform than the TPM, we assume that whenever a TPM is corrupted, then so is the platform.

3. PRELIMINARIES

3.1 Notation

Let $\{0,1\}^\ell$ denote the set of all binary strings of length ℓ. We often switch between integers and their representation as binary strings, e.g., we write $\{0,1\}^\ell$ for the set $[0, 2^\ell - 1]$ of integers. Moreover, we often use $\pm\{0,1\}^\ell$ to denote the set $[-2^\ell + 1, 2^\ell - 1]$.

We need some notation to select the high and low order bits of an integer. Let $\mathtt{LSB}_u(x) := x - 2^u \lfloor \frac{x}{2^u} \rfloor$ and $\mathtt{CAR}_u(x) := \lfloor \frac{x}{2^u} \rfloor$. Let $(x_k \ldots x_0)_b$ denote the binary representation of $x = \sum_{i=0}^{k} 2^i x_i$, e.g., $(1001)_b$ is the binary representation of the integer 9. Then $\mathtt{LSB}_u(x)$ is the integer corresponding to the u least signification bits of (the binary representation of) x, e.g., $\mathtt{LSB}_4(57) = \mathtt{LSB}_4((111001)_b) = (1001)_b = 9$, and $\mathtt{CAR}_u(x)$ is the integer obtained by taking the binary representation of x and right-shifting it by u bits (and cutting of those bits), e.g., $\mathtt{CAR}_4(57) = \mathtt{CAR}_4((111001)_b) = (11)_b = 3$. Also note that $x = \mathtt{LSB}_u(x) + 2^u \mathtt{CAR}_u(x)$.

3.2 Protocols to Prove Knowledge of and Relations among Discrete Logarithms

In our scheme we will use various protocols to prove knowledge of and relations among discrete logarithms. To describe these protocols, we use notation introduced by Camenisch and Stadler [14] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For instance,

$$PK\{(\alpha, \beta, \gamma) : y = g^\alpha h^\beta \ \wedge \ \tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma \ \wedge \ (u \le \alpha \le v)\}$$

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* $u \le \alpha \le v$," 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 of the knowledge 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 [25, 34]. We use the notation $SPK\{(\alpha) : y = g^\alpha\}(m)$ to denote a signature obtained in this way.

3.2.1 Proof of knowledge of the discrete logarithm modulo a composite.

In this paper we apply such PK's and SPK's to the group of quadratic residues modulo a composite n, i.e., $G = \mathrm{QR}_n$, where n is a safe-prime product. This choice for the underlying group has some consequences. First, the protocols are proofs of knowledge under the strong RSA assumption [26]. Second, the largest possible value of the challenge c must be smaller than the smallest factor of G's order. Third, soundness needs special attention in the case that the verifier is not equipped with the factorization of n because then deciding membership in QR_n is believed to be hard. Thus the prover needs to convince the verifier that the elements he presents are indeed quadratic residues, i.e., that the square roots of the presented elements exist. This can in principle be done with a protocol by Fiat and Shamir [25]. However, often it is sufficient to simply execute $PK\{(\alpha) : y^2 = (g^2)^\alpha\}$ or $PK\{(\alpha) : y = \pm g^\alpha\}$ instead of $PK\{(\alpha) : y = g^\alpha\}$. The quantity α is defined as $\log_{g^2} y^2$, which is the same as $\log_g y$ in case y is in QR_n.

3.2.2 Other proofs in a fixed group.

A proof of knowledge of a representation of an element $y \in G$ with respect to several bases $z_1, \ldots, z_v \in G$ [20] is denoted $PK\{(\alpha_1, \ldots, \alpha_v) : y = z_1^{\alpha_1} \cdot \ldots \cdot z_v^{\alpha_v}\}$. A proof of equality of discrete logarithms of two group elements $y_1, y_2 \in G$ to the bases $g \in G$ and $h \in G$, respectively, [19, 21] is denoted $PK\{(\alpha) : y_1 = g^\alpha \wedge y_2 = h^\alpha\}$. Generalizations to prove equalities among representations of the elements $y_1, \ldots, y_v \in G$ to bases $g_1, \ldots, g_v \in G$ are straightforward [14]. A proof of knowledge of a discrete logarithm of $y \in G$ with respect to $g \in G$ such that $\log_g y$ that lies in the integer interval $[a, b]$ is denoted by $PK\{(\alpha) : y = g^\alpha \ \wedge \ \alpha \in [a, b]\}$. Under the strong RSA assumption and if it is assured that the prover is not provided the factorization of the modulus (i.e., is not provided the order of the group) this proof can be done efficiently [4] (it compares to about six ordinary $PK\{(\alpha) : y = g^\alpha\}$.)

3.2.3 Proofs of knowledge of equality in different groups.

The previous protocol can also be used to prove that the discrete logarithms of two group elements $y_1 \in G_1, y_2 \in G_1$ to the bases $g_1 \in G_1$ and $g_2 \in G_2$ in *different* groups G_1 and G_2 are equal [6, 12]. Let the order of the groups be q_1 and q_2, respectively. This proof can be realized only if both discrete logarithms lie in the interval $[0, \min\{q_1, q_2\}]$. The idea is that the prover commits to the discrete logarithm in some group, say $G = \langle g \rangle = \langle h \rangle$ the order of which he does not know, and then executes $PK\{(\alpha, \beta) : y_1 \overset{G_1}{=} g_1^\alpha \ \wedge y_2 \overset{G_2}{=} g_2^\alpha \ \wedge \ C \overset{G}{=} g^\alpha h^\beta \ \wedge \ \alpha \in [0, \min\{q_1, q_2\}]\}$, where C is the commitment. This protocol generalizes to several different groups, to representations, and to arbitrary modular relations.

3.3 Cryptographic Assumptions

We prove security under the strong RSA assumption and the decisional Diffie-Hellman assumption.

Assumption 1 (Strong RSA Assumption). *The strong RSA (SRSA) assumption states that it is computational infeasible, on input a random RSA modulus* n *and a random element* $u \in \mathbb{Z}_n^*$*, to compute values* $e > 1$ *and* v *such that* $v^e \equiv u \pmod{n}$*.*

The tuple (n, u) generated as above is called an *instance* of the *flexible* RSA problem.

Assumption 2 (DDH Assumption). *Let* Γ *be an* ℓ_Γ*-bit prime and* ρ *is an* ℓ_ρ*-bit prime such that* $\rho | \Gamma - 1$*. Let* $\gamma \in \mathbb{Z}_\Gamma^*$ *be an element of order* ρ*. Then,for sufficiently large values of* ℓ_Γ *and* ℓ_ρ*, the distribution* $\{(\delta, \delta^a, \delta^b, \delta^{ab})\}$ *is computationally indistinguishable from the distribution*

$\{(\delta, \delta^a, \delta^b, \delta^c)\}$*, where* δ *is a random element from* $\langle\gamma\rangle$*, and* a*,* b*, and* c *are random elements from* $[0, \rho - 1]$

3.4 The Camenisch-Lysyanskaya Signature Scheme

The direct anonymous attestation scheme is based on the Camenisch-Lysyanskaya (CL) signature scheme [9, 31]. 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, 31]. As we will use somewhat different (and also optimized) protocols for these tasks than those provided in [9, 31], 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 a special RSA modulus $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$. 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.

Message space. Let ℓ_m be a parameter. The message space is the set $\{(m_0, \ldots, m_{L-1}) : m_i \in \pm\{0,1\}^{\ell_m}\}$.

Signing algorithm. On input $m_0, \ldots, m_{L-1}$, 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 [28] 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 $\ell_e > \ell_m + 1$. Also note that the scheme is secure under the strong RSA assumption in the plain model, i.e., does not assume random oracles. If one is to use a hash function to hash messages of arbitrary length to the ℓ_m bits, then the security of the signature scheme further relies on the collision resistance of the used has function. However, note that collision resistant hash functions exist under strong RSA assumption.

4. THE DIRECT ANONYMOUS ATTESTATION SCHEME

4.1 High-Level Description

The basic idea underlying the direct anonymous attestation scheme is similar to the one of the Camenisch-Lysyanskaya anonymous credential system [7, 9, 31] : A trusted hardware module (TPM) chooses a secret "message" f, obtains a Camenisch-Lysyanskaya (CL) signature (aka attestation) on it from the issuer via a secure two-party protocol, and then can convince a verifier that it got attestation anonymously by a proof of knowledge of an attestation. To allow the verifier to recognize rogue TPMs, a TPM must also provide a pseudonym N_V and a proof that the pseudonym is formed correctly, i.e., that it is derived from the TPM's secret f contained in the attestation and a base determined by the verifier. We discuss different ways to handle rogue TPMs later.

Before providing the actual protocols, we first expand on the basic idea and explain some implementation details. First, we split the TPM's secret f into two ℓ_f-bit messages to be signed and denote the (secret) messages by f_0 and f_1 (instead of m_0 and m_1). This split allows for smaller primes e as their size depends on the size of the messages that get signed. Now, the two-party protocol to sign secret messages is as follows (cf. [9]). First, the TPM sends the issuer a commitment to the message-pair (f_0, f_1), i.e., $U := R_0^{f_0} R_1^{f_1} S^{v'}$, where v' is a value chosen randomly by the TPM to "blind" the f_i's. Also, the TPM computes $N_I := \zeta_I^{f_0 + f_1 2^{\ell_f}} \bmod \Gamma$, where ζ_I is a quantity derived from the issuer's name, and sends U and N_I to the issuer. Next, the TPM convinces the issuer that U and N_I are correctly formed (using a proof of knowledge a representation of U w.r.t. the bases R_0, R_1, S and N_I w.r.t. ζ_I) and that the f_i's lie in $\pm\{0,1\}^{\ell_f + \ell_{\mathcal{H}} + \ell_\varnothing + 2}$, where ℓ_f, $\ell_{\mathcal{H}}$, and $\ell_\varnothing$ are security parameters. This interval is larger than the one from which the f_i's actually stem because of the proof of knowledge we apply here. If the issuer accepts the proof, it compares N_I with previous such values obtained from this TPM to decide whether it wants to issue a certificate to TPM w.r.t. N_I or not. (The issuer might not want to grant too many credentials to the TPM w.r.t. different N_I, but should re-grant a credential to a N_I it has already accepted.) To issue a credential, the issuer chooses a random ℓ_v-bit integer v'' and a random ℓ_e-bit prime e, signs the hidden messages by computing $A := \left(\frac{Z}{US^{v''}}\right)^{1/e} \bmod n$, and sends the TPM (A, e, v''). The issuer also proves to the TPM that she computed A correctly. The signature on (f_0, f_1) is then $(A, e, v := v' + v'')$, where v should be kept secret by the TPM (for f_0 and f_1 to remain hidden from the issuer), while A and e can be public.

A TPM can now prove that it has obtained attestation by proving that it got a CL-signature on some values f_0 and f_1. This can be done by a zero-knowledge proof of knowledge of values f_0, f_1, A, e, and v such that $A^e R_0^{f_0} R_1^{f_1} S^v \equiv Z \pmod{n}$. Also, the TPM computes $N_V := \zeta^{f_0 + f_1 2^{\ell_f}} \bmod \Gamma$ and proves that the exponent here is related to those in the attestation, where $\zeta \in \langle\gamma\rangle$, i.e., the subgroup of $\mathbb{Z}_\Gamma^*$ of order ρ.

As mentioned in the introduction, the base ζ is either chosen randomly by the TPM or is generated from a basename value $\mathtt{bsn}_V$ provided by the verifier. The value N_V serves two purposes. The first one is rogue-tagging: If a rogue TPM is found, the values f_0 and f_1 are extracted (how to extract the values f_0 and f_1 is out of the scope of this paper) and put on a blacklist. The verifier can then check N_V against this blacklist by comparing it with $\zeta^{\hat{f}_0 + \hat{f}_1 2^{\ell_f}}$ for all pairs $(\hat{f}_0, \hat{f}_1)$ on the black list. Note that i) the black list can be expected to be short, ii) the exponents $\hat{f}_0 + \hat{f}_1 2^{\ell_f}$ are small (e.g., 200-bits), and iii) batch-verification techniques can be applied [2]; so this check can be done efficiently. Also note that the blacklist need not be certified, e.g., by a certificate revocation agency: whenever f_0, f_1, A, e, and v are discovered, they can be published and everyone can verify whether (A, e, v) is a valid signature on f_0 and f_1. The second purpose of N_V is its use as a pseudonym, i.e., if ζ is not chosen randomly by the TPM but generated from a basename then, whenever the same basename $\texttt{bsn}_V$ is used, the TPM will provide the same value for N_V. This allows the verifier to link different transactions made by the same TPM while not identifying it, and to possibly reject a N_V if it appeared too many times. By defining how often a different basename is used (e.g., a different one per verifier and day), one obtains the full spectrum from full identification via pseudonymity to full anonymity. The way the basename is chosen,the frequency it is changed, and the threshold on how many times a particular N_V can appear before a verifier should reject it, is a question that depends on particular scenarios/applications and is outside of the scope of this document.

We finally note that, whenever possible, multiple proofs by a party are merged in to a single one. Furthermore, the Fiat-Shamir heuristic [25] is used to turn a proof into a "signature of knowledge".

As mentioned before, some operations of the TPM can be outsourced to the host in which the TPM is embedded. In particular, operation that are related to hiding the TPM's identity but not to the capability of producing a proof-signature of knowledge of an attestation can be performed on the host. The rationale behind this is that the host can always reveal a TPM's identity.

Together with their signature scheme, Camenisch and Lysyanskaya also provided protocols to obtain a signature on hidden messages and to prove possession of a certificate [8, 31]. The differences to our join protocol and the sign procedure (which is derived from the protocol to prove possession of a certificate using the Fiat-Shamir heuristic) is that 1) the certificate receiver is shared between the TPM and the host and 2) the protocol to obtain a signature on a hidden message is modified such that it is no longer required that the signer convinces the other parties that the modulus n is the product of two safe prime. The latter is important in practice as proving this property would be rather inefficient and we believe that the way we achieve this is of separate interest, i.e., it can be applied to, e.g., the Ateniese et al. group signature scheme and the Camenisch-Lysyanskaya credential system. The former difference stems from the necessity to outsource as many operations as possible from the TPM to the host. We provide more details about these differences after we have described the respective procedures.

4.2 Security Parameters

We employ the security parameters ℓ_n, ℓ_f, ℓ_e, ℓ'_e, ℓ_v, $\ell_\varnothing$, $\ell_\mathcal{H}$, ℓ_r, ℓ_Γ, and ℓ_ρ, where ℓ_n (2048) is the size of the RSA modulus, ℓ_f (104) is the size of the f_i's (information encoded into the certificate), ℓ_e (368) is the size of the e's (exponents, part of certificate), ℓ'_e (120) is the size of the interval that the e's are chosen from, ℓ_v (2536) is the size of the v's (random value, part of certificate), $\ell_\varnothing$ (80) is the security parameter controlling the statistical zero-knowledge property, $\ell_\mathcal{H}$ (160) is the output length of the hash function used for the Fiat-Shamir heuristic, ℓ_r (80) is the security parameter needed for the reduction in the proof of security, ℓ_Γ (1632) is the size of the modulus Γ, and ℓ_ρ (208) is the size of the order ρ of the sub group of $\mathbb{Z}_\Gamma^*$ that is used for rogue-tagging (the numbers in parentheses are our proposal for these parameters). We require that:

$$\ell_e > \ell_\varnothing + \ell_\mathcal{H} + \max\{\ell_f + 4\ ,\ell'_e + 2\}\ ,$$

$$\ell_v > \ell_n + \ell_\varnothing + \ell_\mathcal{H} + \max\{\ell_f + \ell_r + 3\ ,\ \ell_\varnothing + 2\}, \text{ and}$$

$$\ell_\rho = 2\ell_f .$$

The parameters ℓ_Γ and ℓ_ρ should chosen such that the discrete logarithm problem in the subgroup of $\mathbb{Z}_\Gamma^*$ of order ρ with Γ and ρ being primes such that $2^{\ell_\rho} > \rho > 2^{\ell_\rho - 1}$ and $2^{\ell_\Gamma} > \Gamma > 2^{\ell_\Gamma - 1}$, has about the same difficulty as factoring ℓ_n-bit RSA moduli (see [30]).

Finally, let $\mathcal{H}$ be a collision resistant hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{\ell_\mathcal{H}}$.

4.3 Setup for the Issuer

This section describes how the issuer chooses its public key and secret key. The key generation also produces a non-interactive proof (using the Fiat-Shamir heuristic) that the keys were chosen correctly. The latter will guarantee the security requirements of the host (resp., its user), i.e., that privacy and anonymity of signatures will hold.

1. The issuer chooses a RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$ such that p, p', q, q' are all primes and n has ℓ_n bits. We refer to [23] for an efficient algorithm to select such a modulus.

2. Furthermore, it chooses a random generator g' of QR_n (the group of quadratic residues modulo n).

3. Next, it chooses random integers $x_0, x_1, x_z, x_s, x_h, x_g \in [1, p'q']$ and computes

$$g := {g'}^{x_g} \bmod n,\ h := {g'}^{x_h} \bmod n,\ S := h^{x_s} \bmod n,$$
$$Z := h^{x_z} \bmod n,\ R_0 := S^{x_0} \bmod n,\ R_1 := S^{x_1} \bmod n.$$

4. It produces a non-interactive proof *proof* that R_0, R_1, S, Z, g, and h are computed correctly, i.e., that $g, h \in \langle g' \rangle$, $S, Z \in \langle h \rangle$, and $R_0, R_1 \in \langle S \rangle$. We refer to Appendix A for the details of this proof.

5. It generates a group of prime order: Choose random primes ρ and Γ such that $\Gamma = r\rho + 1$ for some r with $\rho \nmid r$, $2^{\ell_\Gamma - 1} < \Gamma < 2^{\ell_\Gamma}$, and $2^{\ell_\rho - 1} < \rho < 2^{\ell_\rho}$. Choose a random $\gamma' \in_R \mathbb{Z}_\Gamma^*$ such that ${\gamma'}^{(\Gamma-1)/\rho} \not\equiv 1 \pmod{\Gamma}$ and set $\gamma := {\gamma'}^{(\Gamma-1)/\rho} \bmod \Gamma$.

6. Finally, it publishes the public key $(n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$ and the proof *proof* and stores $p'q'$ as its secret key.

Let $H_\Gamma(\cdot)$ and $H(\cdot)$ be two collision resistant hash functions $H_\Gamma(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^{\ell_\Gamma + \ell_\varnothing}$ and $H(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^{\ell_\mathcal{H}}$.

4.4 Verification of the Issuer's Public Key

An issuer's public key can be verified as follows.

1. Verify the proof *proof* that $g, h \in \langle g' \rangle$, $S, Z \in \langle h \rangle$, and $R_0, R_1 \in \langle S \rangle$ as stated in Appendix A.

2. Check whether Γ and ρ are primes, $\rho \mid (\Gamma - 1)$, $\rho \nmid \frac{\Gamma-1}{\rho}$, and $\gamma^\rho \equiv 1 \pmod{\Gamma}$.

3. Check whether all public key parameter have the required length.

If R_0, R_1, S, Z, g, and h are not formed correctly, it could potentially mean that the security properties for the TPM/host do not hold. However, it is sufficient if the platform/host (i.e., owner of the TPM) verifies the proof that R_0, R_1, g, and h are computed correctly once. In principle, it is just sufficient if one representative of platform users checks this proof. Also, if γ does not generate a subgroup of $\mathbb{Z}_\Gamma^*$, the issuer could potentially use this to link different signatures.

As we shall see, it is not important for the security of the platform (i.e., the anonymity/pseudonymity properties) that n is a product of two safe primes.

4.5 Join Protocol

Let $PK_I := (n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$ be the public key of the issuer and let PK'_I be a long-term public key of the issuer used to authenticate PK_I for the DAA. Let $\zeta_I \equiv (H_\Gamma(1\|\mathtt{bsn}_I))^{(\Gamma-1)/\rho} \bmod \Gamma$, where $\mathtt{bsn}_I$ is the issuers long-term basename.

We assume that, prior to running the Join protocol, the TPM and host both verify that PK_I is authenticated by PK'_I.

Let DAAseed be the secret seed used by the TPM in the computation the f_0 and f_1. Note that the reason of using DAAseed instead of generating f_0 and f_1 from a unique random number every time is that it is required to store a single secret value only inside of the TPM for the DAA scheme. Furthermore, let cnt be the current value of the counter keeping track of the number of times that the TPM has run the Join protocol. However, alternatively, the TPM is allowed to re-run the Join protocol with the same cnt value many times. In that case both, the number of times that the TPM has run the Join protocol to re-certify the same f_0 and f_1 pair and the number of times that the TPM has run the Join protocol to get certificates for different f_0 and f_1 pairs, could be traced.

We assume that the TPM and the issuer have established a one-way authentic channel, i.e., the issuer needs to be sure that it talks to the right TPM. Note that authenticity of the channel is enough, i.e., we do not require secrecy, in fact we even assume the host reads all messages and may choose not to forward some messages sent by the issuer to the TPM. In fact, it is sufficient that the value U be transmitted authentically from the TPM to the issuer. In setting of TCG, one can achieve this using the Endorsement Key (EK) pair that was issued to the TPM (see Appendix B).

We are now ready to describe the Join protocol which is provided in Figure 1. As a result of the protocol, the TPM will have obtained secret values f_0, f_1, and v, the host will have values A and e, and the issuer will have values N_I such that $A^e R_0^{f_0} R_1^{f_1} S^v \equiv Z \pmod n$ and $N_I \equiv \zeta_I^{f_0+f_1 2^{\ell_f}} \pmod \Gamma$ holds (cf. §4.1).

Our protocol differs from the one provided in [9] to sign a committed message mainly in that our protocol requires the issuer to prove that A lies in $\langle h \rangle$. The host can conclude that $A \in \langle h \rangle$ from the proof the issuer provides in Step 6, the fact that $A^e U S^{v''} \equiv Z \pmod n$, and the proofs the issuer provides as part of the setup that $S, R_0, R_1, Z \in \langle h \rangle$. We refer to the security proof of Theorem 2 provided in the full paper for the details of this. The reason for requiring $A \in \langle h \rangle$ is to assure that later, in the signing algorithm, A can be statistically hidden in $\langle h \rangle$. Otherwise, an adversarial signer could compute A for instance as $b\big(\frac{Z}{US^{v''}}\big)^{1/e} \bmod n$ using some b such that $b^e = 1$ and $b \notin \langle h \rangle$. As a DAA-signature contains the value $T_1 = Ah^w$ for a random w (see DAA-signing protocol), and adversarial signer would be able to link T_1 to A, e.g., by testing $T_1 \in \langle h \rangle$. Prior schemes such as [1, 7, 9] have prevented this by ensuring that n is a safe-prime product and then made sure that all elements are cast into QR_n. However, proving that a modulus is a safe-prime product is rather inefficient [11] and hence the setup of these schemes is not really practical. We note that the proof in Step 6 is zero-knowledge only if the issuer has chosen n as a safe-prime product. This is a property that the issuer is interested in, and not the TPM, host, or users of a platform.

Because our security proof requires rewinding to extract f_0, f_1, and v' from an adversarial TPM, the join protocol can only be run sequentially, i.e., not in parallel with many TPMs. At some loss of efficiency, this drawback could be overcome for instance by using the verifiable encryption [13] of these values.

4.6 DAA-Signing Protocol

We now describe how a platform can prove that is got an anonymous attestation credential and at the same time authenticate a message. Thus, the platform gets as input a message m to DAA-sign. We remark that in some cases the message m will be generated by the TPM and might not be known to the host. That is, the TPM signs an Attestation Identity Key (AIK), an RSA public key that is has generated, which the TPM will later use to sign its internal registers.

Let $n_v \in \{0,1\}^{\ell_\mathcal{H}}$ be a nonce and $\mathtt{bsn}_V$ a basename value provided by the verifier. Let b be a byte describing the use of the protocol, i.e., $b = 0$ means that the message m is generated by the TPM and $b = 1$ means that the message m was input to the TPM.

We are now ready to describe the DAA-Sign procedure which is a protocol between the TPM and its host and is provided in Figure 2. As a result of the protocol, the host will have obtained a signature $\sigma := (\zeta, (T_1, T_2), N_V, c, n_t, (s_v, s_{f_0}, s_{f_1}, s_e, s_{ee}, s_w, s_{ew}, s_r, s_{er}))$. on the message m.

Let us give some intuition about why this signature should convince a verifier that N_V links to secrets that were certified. The first term in the SPK can be rewritten as $Z \equiv (\pm\frac{T_1}{h^{ew/e}})^e R_0^{f_0} R_1^{f_1} S^v \pmod n$, if e divides we (see proof of security). The second two terms show that e indeed divides we, so the rewriting works. Therefore, because the last terms that show that the f_i's and e satisfy the length requirements, $(\frac{T_1}{h^{ew/e}}, e, v)$ is a valid certificate for f_0 and f_1. The remaining term shows that N_V is based on the same f_0 and f_1.

The main difference of this DAA-signing protocol to the signature generation in prior schemes such as [1, 7, 9] is that here we have distributed the necessary operation to two parties, the TPM and the host. Basically, the TPM only produces the proof-signature $SPK\{(f_0, f_1, v) : (Z/A^e) \equiv R_0^{f_0} R_1^{f_1} S^v \pmod n \ \wedge N_V \equiv \zeta^{f_0+f_1 2^{\ell_f}} \pmod \Gamma)\}(n_t\|n_v\|b\|m)$, which the host then extends to the full DAA-signature (note that (Z/A^e) is known to the host). Note that the SPK produced by the TPM is not anonymous (even with a random ζ) as the value (Z/A^e) would fully identify the TPM. While it is intuitively obvious that the host can-

1. The host computes $\zeta_I := (H_\Gamma(1\|\texttt{bsn}_I))^{(\Gamma-1)/\rho} \bmod \Gamma$ and sends ζ_I to the TPM.

2. The TPM checks whether $\zeta_I^\rho \equiv 1 \pmod{\Gamma}$. Let $i := \lfloor\frac{\ell_\rho+\ell_\varnothing}{\ell_\mathcal{H}}\rfloor$ (i will be 1 for values of the parameters selected in Section 4.2). The TPM computes

$$f := H(H(\texttt{DAAseed}\|H(PK_I'))\|\texttt{cnt}\|0)\|\ldots\|H(H(\texttt{DAAseed}\|H(PK_I'))\|\texttt{cnt}\|i) \pmod{\rho} \ ,$$

$$f_0 := \texttt{LSB}_{\ell_f}(f) \ , \quad f_1 := \texttt{CAR}_{\ell_f}(f) \ , \quad v' \in_R \{0,1\}^{\ell_n+\ell_\varnothing} \ , \quad U := R_0^{f_0}R_1^{f_1}S^{v'} \bmod n \ , \quad N_I := \zeta_I^{f_0+f_1 2^{\ell_f}} \bmod \Gamma$$

   and sends U and N_I to the host who forwards them to the issuer.

3. The issuer checks for all (f_0, f_1) on the rogue list whether $N_I \overset{?}{\not\equiv} (\zeta^{f_0+f_1 2^{\ell_f}}) \pmod{\Gamma}$. The issuer also checks this for the N_I this platform had used previously. If the issuer finds the platform to be rogue, it aborts the protocol.

4. The TPM proves to the issuer knowledge of f_0, f_1, and v': it executes as prover the protocol

$$SPK\{(f_0,f_1,v') : \quad U \equiv \pm R_0^{f_0}R_1^{f_1}S^{v'} \pmod{n} \ \wedge \ N_I \equiv \zeta_I^{f_0+f_1 2^{\ell_f}} \pmod{\Gamma} \ \wedge$$
$$f_0, f_1 \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+2} \ \wedge \ v' \in \{0,1\}^{\ell_n+\ell_\varnothing+\ell_\mathcal{H}+2}\}(n_t\|n_i)$$

   with the issuer as the verifier. This protocol is implemented as follows, where some non-critical operations are performed by the host and not be the TPM.

   (a) The TPM picks random integers $r_{f_0}, r_{f_1} \in_R \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}}$ and $r_{v'} \in_R \{0,1\}^{\ell_n+2\ell_\varnothing+\ell_\mathcal{H}}$, computes $\tilde{U} := R_0^{r_{f_0}}R_1^{r_{f_1}}S^{r_{v'}} \bmod n$ and $\tilde{N}_I := \zeta_I^{r_{f_0}+r_{f_1}2^{\ell_f}} \bmod \Gamma$, and sends $\tilde{U}$ and $\tilde{N}_I$ to the host.

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

   (c) The host computes $c_h := H(n\|R_0\|R_1\|S\|U\|N_I\|\tilde{U}\|\tilde{N}_I\|n_i)$ and sends c_h to the TPM

   (d) The TPM chooses a random $n_t \in \{0,1\}^{\ell_\varnothing}$ and computes $c := H(c_h\|n_t) \in [0, 2^{\ell_\mathcal{H}}-1]$.

   (e) The TPM computes $s_{f_0} := r_{f_0} + c\cdot f_0$, $s_{f_1} := r_{f_1} + c\cdot f_1$, and $s_{v'} := r_{v'} + c\cdot v'$ and sends the host $(c, n_t, s_{f_0}, s_{f_1}, s_{v'})$.

   (f) The host forwards $(c, n_t, s_{f_0}, s_{f_1}, s_{v'})$ to the issuer.

   (g) The issuer verifies the proof by computing $\hat{U} := U^{-c}R_0^{s_{f_0}}R_1^{s_{f_1}}S^{s_{v'}} \bmod n$ and $\hat{N}_I := N_I^{-c}\zeta_I^{s_{f_0}+2^{\ell_f}s_{f_1}} \bmod \Gamma$ and checking if $c \overset{?}{=} H(H(n\|R_0\|R_1\|S\|U\|N_I\|\hat{U}\|\hat{N}_I\|n_i)\|n_t)$, $s_{f_0}, s_{f_1} \overset{?}{\in} \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+1}$, and $s_{v'} \overset{?}{\in} \{0,1\}^{\ell_n+2\ell_\varnothing+\ell_\mathcal{H}+1}$.

5. 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 $v'' := \hat{v} + 2^{\ell_v-1}$ and

$$A := \left(\frac{Z}{US^{v''}}\right)^{1/e} \bmod n \ .$$

6. To convince the host that A was correctly computed, the issuer as prover runs the protocol

$$SPK\{(d) : \quad A \equiv \pm\left(\frac{Z}{US^{v''}}\right)^d \pmod{n}\}(n_h)$$

   with the host:

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

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

$$\tilde{A} := \left(\frac{Z}{US^{v''}}\right)^{r_e} \bmod n \ , \qquad c' := H(n\|Z\|S\|U\|v''\|A\|\tilde{A}\|n_h) \ , \qquad \text{and} \qquad s_e := r_e - c'/e \bmod p'q' \ ,$$

   and sends c', s_e, and (A, e, v'') to the host.

   (c) The host verifies whether e is a prime and lies in $[2^{\ell_e-1}, 2^{\ell_e-1}+2^{\ell_e'-1}]$, computes $\hat{A} := A^{c'}\left(\frac{Z}{US^{v''}}\right)^{s_e} \bmod n$, and checks whether $c' \overset{?}{=} H(n\|Z\|S\|U\|v''\|A\|\hat{A}\|n_h)$.

7. The host forwards v'' to the TPM.

8. The TPM receives v'', sets $v := v'' + v'$, and stores (f_0, f_1, v).

Figure 1: The Join protocol. The inputs to the TPM are $(n, R_0, R_1, S, \rho, \Gamma)$, DAAseed, cnt, $H(\boldsymbol{PK}_I')$, the input to the host is $(n, R_0, R_1, S, Z, \rho, \Gamma)$, and the input to the issuer are $(n, R_0, R_1, S, Z, \rho, \Gamma)$, p and q.

The signing algorithm is as follows.

1. (a) Depending on the verifier's request (i.e., whether $\texttt{bsn}_V \neq \perp$ or not), the host computes ζ as follows

$$\zeta \in_R \langle\gamma\rangle \qquad \text{or} \qquad \zeta := (H_\Gamma(1\|\texttt{bsn}_V))^{(\Gamma-1)/\rho} \bmod \Gamma$$

   and sends ζ to the TPM.

   (b) The TPM checks whether $\zeta^\rho \equiv 1 \pmod{\Gamma}$.

2. (a) The host picks random integers $w, r \in \{0,1\}^{\ell_n+\ell_\varnothing}$ and computes $T_1 := Ah^w \bmod n$ and $T_2 := g^w h^e (g')^r \bmod n$.

   (b) The TPM computes $N_V := \zeta^{f_0+f_1 2^{\ell_f}} \bmod \Gamma$ and sends N_V to the host.

3. Now, the TPM and host together produce a "signature of knowledge" that T_1 and T_2 commit to a certificate and N_V was computed using the secret key going with that certificate. That is, they compute the "signature of knowledge"

$$\begin{aligned} &SPK\{(f_0, f_1, v, e, w, r, ew, ee, er) : \\ &\qquad Z \equiv \pm T_1^e R_0^{f_0} R_1^{f_1} S^v h^{-ew} \pmod{n} \ \wedge\ T_2 \equiv \pm g^w h^e g'^r \pmod{n} \ \wedge \\ &\qquad 1 \equiv \pm T_2^{-e} g^{ew} h^{ee} g'^{er} \pmod{n} \ \wedge\ N_V \equiv \zeta^{f_0+f_1 2^{\ell_f}} \pmod{\Gamma} \ \wedge \\ &\qquad f_0, f_1 \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+2} \ \wedge\ (e - 2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}+1}\}(n_t\|n_v\|b\|m) \ . \end{aligned}$$

   Most of the secrets involved are actually known by the host; in fact only the values involving f_0, f_1, and v need to be computed by the TPM, as the reader can see below.

   (a) i. The TPM picks random integers $r_v \in_R \{0,1\}^{\ell_v+\ell_\varnothing+\ell_\mathcal{H}}$ and $r_{f_0}, r_{f_1} \in_R \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}}$ and computes

$$\tilde{T}_{1t} := R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_v} \bmod n \qquad \tilde{r}_f := r_{f_0} + r_{f_1} 2^{\ell_f} \bmod \rho \qquad \tilde{N}_V := \zeta^{\tilde{r}_f} \bmod \Gamma \ .$$

   The TPM sends $\tilde{T}_{1t}$ and $\tilde{N}_V$ to the host.

   ii. The host picks random integers

$$\begin{aligned} r_e &\in_R \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}} \ , & r_{ee} &\in_R \{0,1\}^{2\ell_e+\ell_\varnothing+\ell_\mathcal{H}+1} \ , \\ r_w, r_r &\in_R \{0,1\}^{\ell_n+2\ell_\varnothing+\ell_\mathcal{H}} \ , & r_{ew}, r_{er} &\in_R \{0,1\}^{\ell_e+\ell_n+2\ell_\varnothing+\ell_\mathcal{H}+1} \end{aligned}$$

   and computes

$$\tilde{T}_1 := \tilde{T}_{1t} T_1^{r_e} h^{-r_{ew}} \bmod n \ , \qquad \tilde{T}_2 := g^{r_w} h^{r_e} g'^{r_r} \bmod n \ , \qquad \tilde{T}'_2 := T_2^{-r_e} g^{r_{ew}} h^{r_{ee}} g'^{r_{er}} \bmod n \ .$$

   (b) i. Host computes

$$c_h := H((n\|g\|g'\|h\|R_0\|R_1\|S\|Z\|\gamma\|\Gamma\|\rho)\|\zeta\|(T_1\|T_2)\|N_V\|(\tilde{T}_1\|\tilde{T}_2\|\tilde{T}'_2)\|\tilde{N}_V)\|n_v) \in [0, 2^{\ell_\mathcal{H}} - 1] \ .$$

   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 host.

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

$$s_v := r_v + c \cdot v \ , \qquad s_{f_0} := r_{f_0} + c \cdot f_0 \ , \quad \text{and} \qquad s_{f_1} := r_{f_1} + c \cdot f_1$$

   and sends (s_v, s_{f_0}, s_{f_1}) to the host.

   ii. The host computes (over the integers)

$$\begin{aligned} s_e &:= r_e + c \cdot (e - 2^{\ell_e - 1}) \ , & s_{ee} &:= r_{ee} + c \cdot e^2 \ , & s_w &:= r_w + c \cdot w \ , \\ s_{ew} &:= r_{ew} + c \cdot w \cdot e \ , & s_r &:= r_r + c \cdot r \ , & s_{er} &:= r_{er} + c \cdot e \cdot r \ . \end{aligned}$$

4. The host outputs the signature $\sigma := (\zeta, (T_1, T_2), N_V, c, n_t, (s_v, s_{f_0}, s_{f_1}, s_e, s_{ee}, s_w, s_{ew}, s_r, s_{er}))$.

Figure 2: The DAA-Signing protocol. The input to the protocol for the TPM is m, $(n, R_0, R_1, S, \Gamma, \rho)$, and (f_0, f_1, v), and the host's input to the protocol is m, the certificate (A, e) and $(n, g, g', h, R_0, R_1, S, Z, \gamma, \Gamma, \rho)$.

not generate such signatures on its own or turn one by a TPM into one on a different message, we provide a formal proof of this in Section 5.

4.7 Verification Algorithm

The verification algorithm is provided in Figure 3. The check $N_V, \zeta \overset{?}{\in} \langle\gamma\rangle$ can be done by raising N_V and ζ to the order of γ (which is ρ) and checking whether the result is 1. In case ζ is random, one can apply so called batch verification techniques (cf. [2]) to obtain a considerable speed-up of the verification step 4. Also note that the involved exponents are relatively small. Finally, if ζ is not random, one could precompute $\zeta^{f_0+f_1 2^{\ell_f}}$ for all (f_0, f_1) on the rogue list.

4.8 On Rogue Tagging

When a certificate (A, e, v) and values f_0 and f_1 are found (e.g., on the Internet or embedded into some software), they should be distributed to all potential verifiers. These verifiers can then check whether $A^e R_0^{f_0} R_1^{f_1} S^v \equiv Z \pmod{n}$ holds and then put f_0 and f_1 on their list of rogue keys. Note that this does not involve a certificate revocation authority.

4.9 Inputs and Outputs of the Parties

To actually meet the specification of DAA we gave in Section 2, it remains to define the outputs of the parties in the real system. The hosts do not output anything. The issuer outputs the identity of a platform after it has joined; if a platform that has been tagged rogue tries to join the protocols, it outputs the identity of this platform and the relevant counter value. Finally, when a verifier sees a valid signature (this excludes signatures generated by a rogue TPM), it produces its output as follows: (1) If the signature was produced by a rogue TPM, it outputs m together with the note that m was signed by a rogue. (2) If the signature was produced w.r.t. $\texttt{bsn}_V = \perp$, it just outputs the message. (3) If the signature was produced with a ζ derived from a basename (i.e., w.r.t. $\texttt{bsn}_V \neq \perp$), it looks in its database whether it already assigned a pseudonym P to the pair (ζ, N_V); otherwise it chooses new random pseudonym $P \in_R \{0,1\}^{\ell_\sigma}$ and assigns it the pair (the quantity ℓ_σ is a security parameter). In either case, it outputs P and m.

4.10 Performance Analysis

Let us consider the amount of computations the individual parties have to perform in the "join", "DAA-sign", and "verify" procedures. The computations in all these procedure are dominated by the exponentiations, so it is sufficient to consider only these. Also note that in a good implementation, a multi-base exponentiation is not much more expensive that a single-base exponentiation. As this requires more memory that available on a TPM, a multi-base exponentiation needs implemented as multiple single-base exponentiation in this case. We thus count multi-based exponentiation that have to be performed by the TPM as the multiple single-base exponentiation and for all other parties as a single single-base exponentiation.

4.10.1 Setup for the Issuer

To generate a public key, the issuer has to produce an RSA modulus that is the product of two safe primes. If the modulus should be about 2048 bits, that means that one has to try about two times 30 random numbers and check whether they are safe primes by using for instance trial division and the variant of the Miller-Rabin test provided by Cramer and Shoup [23]. The generation of the proof that the other parameter in the issuer's key are correct requires 6 times 160 exponentiations modulo a 2048 bit composite, for security parameter $\ell_{\mathcal{H}}$ of 160. Here one can use Chinese remaindering to speed up these computations. The verification of the proof requires the same number of exponentiations; however, Chinese remaindering is not applicable here. Note that such the verification needs to be performed only once and not necessarily by every user of the system (e.g., only be a user representative). Moreover, if some RSA modulus was available whose factorization is not known to the issuer, this proof could be sped up such that only about 12 exponentiations would be required for its generation and verification. We do not pursue this issue further here.

4.10.2 Join Protocol

Here, the TPM needs to perform 6 exponentiations modulo n (2048 bits) and 3 exponentiations modulo Γ (1660 bits), the host 1 exponentiation modulo n and 1 exponentiation modulo Γ, and the issuer 3 exponentiations modulo n (here Chinese remaindering can be used) and 1 exponentiation modulo Γ.

4.10.3 DAA-Sign and Verification

To generate a DAA-signature, the TPM needs to perform 1 exponentiation modulo n (2048 bits) and 3 exponentiations modulo Γ (1660 bits), the host 5 exponentiations modulo n and 1 exponentiation modulo Γ. If the basename is known, all exponentiations on the TPM and the host can be precomputed; otherwise all exponentiations modulo n on the host and the TPM can be precomputed.

To verify a signature, 3 exponentiations modulo n and 1 exponentiation modulo Γ.

4.10.4 Experimental Results

A prototype of direct anonymous attestation has been implemented at IBM Research in Zurich in the JAVA programming language. That is, all the parties, in particular also the TPM, were implemented in software. For all group operations, the standard JAVA BigInteger class was used. That is, a multi-base exponentiation was implemented as several exponentiations. All the protocols and algorithms were implemented as single threaded programs. In particular, no parallel computations by the parties was realized (e.g., the join and the sign protocol would allow the host to perform some computations while waiting for a response from the TPM or the issuer) and no precomputation was implemented.

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 join protocol required 2.4 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 host, and about 50% was used by the issuer. The sign protocol required 4.4 seconds of running time in total (excluding communication time), about 8% of the time was used by the TPM, about 47% was used by the host, and about 45% was used by the verifier.

The verification time of a DAA signature can be significantly reduced by better exponentiations algorithms that

A signature

$$\sigma = (\zeta, (T_1, T_2), N_V, c, n_t, (s_v, s_{f_0}, s_{f_1}, s_e, s_{ee}, s_w, s_{ew}, s_r, s_{er}))$$

on a message m w.r.t. the public key $(n, g, g', h, R_0, R_1, S, Z, \gamma, \Gamma, \rho)$ is as verified as follows.

1. Compute

$$\hat{T}_1 := Z^{-c} T_1^{s_e + c2^{\ell_e - 1}} R_0^{s_{f_0}} R_1^{s_{f_1}} S^{s_v} h^{-s_{ew}} \bmod n \ , \qquad \hat{T}_2 := T_2^{-c} g^{s_w} h^{s_e + c2^{\ell_e - 1}} g'^{s_r} \bmod n \ ,$$

$$\hat{T}'_2 := T_2^{-(s_e + c2^{\ell_e - 1})} g^{s_{ew}} h^{s_{ee}} g'^{s_{er}} \bmod n \ , \quad \text{and} \qquad \hat{N}_V := N_V^{-c} \zeta^{s_{f_0} + s_{f_1} 2^{\ell_f}} \bmod \Gamma \ .$$

2. Verify that

$$c \overset{?}{=} H(H(H((n \| g \| g' \| h \| R_0 \| R_1 \| S \| Z \| \gamma \| \Gamma \| \rho) \| \zeta \| (T_1 \| T_2) \| N_V \| (\hat{T}_1 \| \hat{T}_2 \| \hat{T}'_2) \| \hat{N}_V \| n_v) \| n_t) \| b \| m) \ ,$$

$$N_V, \zeta \overset{?}{\in} \langle \gamma \rangle \ , \quad s_{f_0}, s_{f_1} \overset{?}{\in} \{0,1\}^{\ell_f + \ell_\varnothing + \ell_\mathcal{H} + 1} \ , \quad \text{and} \quad s_e \overset{?}{\in} \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_\mathcal{H} + 1} \ .$$

3. If ζ was derived from a verifier's basename, check whether $\zeta \overset{?}{\equiv} (H_\Gamma(1 \| \mathtt{bsn}_V))^{(\Gamma - 1)/\rho} \pmod{\Gamma}$.

4. For all (f_0, f_1) on the rogue list check whether $N_V \overset{?}{\not\equiv} (\zeta^{f_0 + f_1 2^{\ell_f}}) \pmod{\Gamma}$.

Figure 3: The verification algorithm

use precomputed base values. Also, as mentioned earlier, all (expensive) signing operations on the host can be precomputed as can some on the TPM.

5. SECURITY PROOFS

The following theorem establishes the security of our scheme.

THEOREM 2. *The protocols provided in Section 4 securely implement a secure direct anonymous attestation system under the decisional Diffie-Hellman assumption in $\langle \gamma \rangle$ and the strong RSA assumption in the random oracle model.*

We refer to full version of this paper for the actual security proofs and give here only a very high-level overview of the proof. The proof of Theorem 2 consists of the description of a simulator and arguments that the environment cannot distinguish whether it is run in the real system, interacting with $\mathcal{A}$ and the real parties, or in the ideal system, interacting with $\mathcal{S}$ and the ideal parties. Recall that the simulator interacts with $\mathcal{T}$ on behalf of the corrupted parties of the ideal system, and simulates the real-system adversary $\mathcal{A}$ towards the environment $\mathcal{E}$.

The simulator, which has black box access to the adversary, basically runs the real system protocol in the same way as an honest party would, apart from the DAA-signing protocol, where the simulator is just told by $\mathcal{T}$ that some party signed a message (possible w.r.t. a pseudonym) but it does not know which party signed. Thus the simulator just chooses a random N_V and then forges a signature by using the zero-knowledge simulator of the *SPK* proof and the power over the random oracle. Also, if the simulator notes that the adversary signed some message, it chooses some corrupted host and tells $\mathcal{T}$ that this host has signed on behalf of a party. The simulator will fail in cases where the adversary manages to forge a signature, to sign on behalf of a honest TPM/host, or to tag an honest TPM as a rogue. We show that these cases cannot occur only under the strong RSA and the discrete logarithm assumption. We then show that if the simulator does not fail then, under the decisional Diffie-Hellman assumption, the environment will not be able to tell whether or not it is run in the real system interacting with the adversary or the ideal system interacting with the simulator.

6. CONCLUSION

The protocols we describe could be extended in many ways. For instance, only minor changes would be necessary to allow the issuer to use any RSA modulus instead of only safe-prime products. However, one would need to make a small order assumption [24]. Another extension would guarantee anonymity/pseudonymity to the host (in fact to its user) even if the TPM deviates from the protocol (cf. [21]). Finally, as we have already mentioned in Section 4.5, one could extend the join protocol in such a way that the system can also be proved secure if issuer runs the protocol concurrently with different TPMs.

7. ACKNOWLEDGEMENT

The authors are grateful to Dave Challener, Graeme Proudler, Micheal Waidner, and Jim Ward for the various lengthy discussions we had about this proposal. Roger Zimmermann implemented this scheme in JAVA and provided valuable feedback. Caroline Kudla, Anna Lysyanskaya, John Malone-Lee, Wenbo Mao and many others gave us helpful comments on the earlier versions of the paper.

8. REFERENCES

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

[2] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *LNCS*, pages 236–250. Springer Verlag, 1998.

[3] D. Boneh, E. Brickell, L. Chen, and H. Shacham. Set signatures. Manuscript, 2003.

[4] F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444. Springer Verlag, 2000.

[5] E. Brickell. An efficient protocol for anonymously providing assurance of the container of a private key. Submitted to the Trusted Computing Group, Apr. 2003.

[6] E. F. Brickell, D. Chaum, I. B. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret. In C. Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *LNCS*, pages 156–166. Springer-Verlag, 1988.

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

[8] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer Verlag, 2002.

[9] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In S. Cimato, C. Galdi, and G. Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002*, volume 2576 of *LNCS*, pages 268–289. Springer Verlag, 2003.

[10] J. Camenisch and M. Michels. A group signature scheme with improved efficiency. In K. Ohta and D. Pei, editors, *Advances in Cryptology — ASIACRYPT '98*, volume 1514 of *LNCS*, pages 160–174. Springer Verlag, 1998.

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

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

[13] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144, 2003.

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

[15] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

[16] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[17] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology — Proceedings of CRYPTO '82*, pages 199–203. Plenum Press, 1983.

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

[19] D. Chaum. Zero-knowledge undeniable signatures. In I. B. Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *LNCS*, pages 458–464. Springer-Verlag, 1991.

[20] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In D. Chaum and W. L. Price, editors, *Advances in Cryptology — EUROCRYPT '87*, volume 304 of *LNCS*, pages 127–141. Springer-Verlag, 1988.

[21] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer-Verlag, 1993.

[22] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *LNCS*, pages 257–265. Springer-Verlag, 1991.

[23] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

[24] I. Damgård and M. Koprowski. Practical threshold RSA signatures without a trusted dealer. In B. Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 152–165. Springer Verlag, 2001.

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

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

[27] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, pages 123–139. Springer Verlag, 1999.

[28] 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.

[29] J. Kilian and E. Petrank. Identity escrow. In H. Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1642 of *LNCS*, pages 169–185, Berlin, 1998. Springer Verlag.

[30] A. K. Lenstra and E. K. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[31] A. Lysyanskaya. *Signature schemes and applications to cryptographic protocol design*. PhD thesis, Massachusetts Institute of Technology, Sept. 2002.

[32] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254. ACM press, Nov. 2000.

[33] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 184–200. IEEE Computer Society, IEEE Computer Society Press, 2001.

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

[35] 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.

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

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

APPENDIX

A. GENERATION AND VERIFICATION OF A PROOF THAT PUBLIC KEY ELEMENTS LIE IN THE RIGHT SUBGROUPS

This section describes the steps to be performed by the issuer to prove that it computed R_0, R_1, S, Z, g, and h correctly, i.e., that $g, h \in \langle g' \rangle$, $S, Z \in \langle h \rangle$, and $R_0, R_1 \in \langle S \rangle$. It also describes the steps for a host to verify the proof and thus to establish that the privacy and anonymity properties of the scheme will hold.

Figure 4 presents the algorithm to generate this proof while Figure 5 presents the algorithm to verify it. The proof uses binary challenges (the c_i's) and it is easy to see that it is indeed a zero-knowledge proof that R_0 and R_1 lie in $\langle S \rangle$, that Z and S lie in $\langle h \rangle$, and that g and h lie in $\langle g' \rangle$. We note that if some RSA modulus was available whose factorization is not known to the issuer, this proof could done using non-binary challenges which would make it about $\ell_{\mathcal{H}}$ times more efficient. We do not persue this issue furhter here.

B. AUTHENTICATING A TPM W.R.T. AN ENDORSEMENT KEY

In the join protocol, the issuer must be sure that the value U stems from the TPM that owns a given endorsement public key EK. In this paper we just assume that the issuer receives U in an authentic manner, the following protocol could be used to achieve this.

(a) The issuer chooses a random $n_e \in \{0,1\}^{\ell_\varnothing}$ and encrypts n_e under the EK and sends the encryption to the TPM.

(b) The TPM decrypts this and thereby retrieves some string n_e. Then, the TPM computes $a_U := H(U \| n_e)$ and sends a_U to the issuer.

(c) The issuer verifies if $a_U = H(U \| n_e)$ holds.

This protocol should be executed between the steps 2 and 4 of the join protocol. This protocol is in the same spirit as the protocol to "authenticate" the AIK in the TCG TPM 1.1b specification.

1. The prover chooses random

$$\tilde{x}_{(g,1)},\ldots,\tilde{x}_{(g,\ell_{\mathcal{H}})} \in_R [1,p'q'] \qquad \tilde{x}_{(h,1)},\ldots,\tilde{x}_{(h,\ell_{\mathcal{H}})} \in_R [1,p'q'] \qquad \tilde{x}_{(s,1)},\ldots,\tilde{x}_{(s,\ell_{\mathcal{H}})} \in_R [1,p'q']$$
$$\tilde{x}_{(z,1)},\ldots,\tilde{x}_{(z,\ell_{\mathcal{H}})} \in_R [1,p'q'] \qquad \tilde{x}_{(0,1)},\ldots,\tilde{x}_{(0,\ell_{\mathcal{H}})} \in_R [1,p'q'] \qquad \tilde{x}_{(1,1)},\ldots,\tilde{x}_{(1,\ell_{\mathcal{H}})} \in_R [1,p'q']$$

   and computes

$$\tilde{g}_{(g,i)} := {g'}^{\tilde{x}_{(g,i)}} \bmod n \qquad \tilde{h}_{(h,i)} := {g'}^{\tilde{x}_{(h,i)}} \bmod n \qquad \tilde{S}_{(s,i)} := h^{\tilde{x}_{(s,i)}} \bmod n$$
$$\tilde{Z}_{(z,i)} := h^{\tilde{x}_{(z,i)}} \bmod n \qquad \tilde{R}_{(0,i)} := S^{\tilde{x}_{(0,i)}} \bmod n \qquad \tilde{R}_{(1,i)} := S^{\tilde{x}_{(1,i)}} \bmod n$$

   for $i = 1,\ldots,\ell_{\mathcal{H}}$.

2. The prover computes

$$c := H(n, g', g, h, S, Z, R_0, R_1, \tilde{g}_{(g,1)},\ldots,\tilde{g}_{(g,\ell_{\mathcal{H}})}, \tilde{h}_{(h,1)},\ldots,\tilde{h}_{(h,\ell_{\mathcal{H}})},$$
$$\tilde{S}_{(s,1)},\ldots,\tilde{S}_{(s,\ell_{\mathcal{H}})}, \tilde{Z}_{(z,1)},\ldots,\tilde{Z}_{(z,\ell_{\mathcal{H}})}, \tilde{R}_{(0,1)},\ldots,\tilde{R}_{(0,\ell_{\mathcal{H}})}, \tilde{R}_{(1,1)},\ldots,\tilde{R}_{(1,\ell_{\mathcal{H}})}) \ .$$

3. The prover computes

$$\hat{x}_{(g,i)} := \tilde{x}_{(g,i)} - c_i x_g \bmod p'q' \qquad \hat{x}_{(h,i)} := \tilde{x}_{(h,i)} - c_i x_h \bmod p'q' \qquad \hat{x}_{(s,i)} := \tilde{x}_{(s,i)} - c_i x_s \bmod p'q'$$
$$\hat{x}_{(z,i)} := \tilde{x}_{(z,i)} - c_i x_z \bmod p'q' \qquad \hat{x}_{(0,i)} := \tilde{x}_{(0,i)} - c_i x_0 \bmod p'q' \qquad \hat{x}_{(1,i)} := \tilde{x}_{(1,i)} - c_i x_1 \bmod p'q'$$

   for $i = 1,\ldots,\ell_{\mathcal{H}}$, where c_i is the i-th bit of c.

4. The prover publishes

$$\mathit{proof} := (c, \hat{x}_{(g,1)},\ldots,\hat{x}_{(g,\ell_{\mathcal{H}})}, \hat{x}_{(h,1)},\ldots,\hat{x}_{(h,\ell_{\mathcal{H}})},$$
$$\hat{x}_{(s,1)},\ldots,\hat{x}_{(s,\ell_{\mathcal{H}})}, \hat{x}_{(z,1)},\ldots,\hat{x}_{(z,\ell_{\mathcal{H}})}, \hat{x}_{(0,1)},\ldots,\hat{x}_{(0,\ell_{\mathcal{H}})}, \hat{x}_{(1,1)},\ldots,\hat{x}_{(1,\ell_{\mathcal{H}})},) \ .$$

   as proof that $g, h \in \langle g' \rangle$, $S, Z \in \langle h \rangle$, and $R_0, R_1 \in \langle S \rangle$.

Figure 4: Generation of the proof that the parameter in the issuer's public key were correctly generated.

A proof

$$\mathit{proof} = (c, \hat{x}_{(g,1)},\ldots,\hat{x}_{(g,\ell_{\mathcal{H}})}, \hat{x}_{(h,1)},\ldots,\hat{x}_{(h,\ell_{\mathcal{H}})}, \hat{x}_{(s,1)},\ldots,\hat{x}_{(s,\ell_{\mathcal{H}})}, \hat{x}_{(z,1)},\ldots,\hat{x}_{(z,\ell_{\mathcal{H}})},$$
$$\hat{x}_{(0,1)},\ldots,\hat{x}_{(0,\ell_{\mathcal{H}})}, \hat{x}_{(1,1)},\ldots,\hat{x}_{(1,\ell_{\mathcal{H}})})$$

that R_0, R_1, S, Z, g, and h are correctly formed is verified as follows

1. Compute

$$\hat{g}_{(g,i)} := g^{c_i} {g'}^{\hat{x}_{(g,i)}} \bmod n \qquad \hat{h}_{(h,i)} := h^{c_i} {g'}^{\hat{x}_{(h,i)}} \bmod n \qquad \hat{S}_{(s,i)} := S^{c_i} h^{\hat{x}_{(s,i)}} \bmod n$$
$$\hat{Z}_{(z,i)} := Z^{c_i} h^{\hat{x}_{(z,i)}} \bmod n \qquad \hat{R}_{(0,i)} := R_0^{c_i} S^{\hat{x}_{(0,i)}} \bmod n \qquad \hat{R}_{(1,i)} := R_1^{c_i} S^{\hat{x}_{(1,i)}} \bmod n$$

   for $i = 1,\ldots,\ell_{\mathcal{H}}$, where c_i is the i-th bit of c.

2. Verify

$$c \overset{?}{=} H(n, g', g, h, S, Z, R_0, R_1, \hat{g}_{(g,1)},\ldots,\hat{g}_{(g,\ell_{\mathcal{H}})}, \hat{h}_{(h,1)},\ldots,\hat{h}_{(h,\ell_{\mathcal{H}})},$$
$$\hat{S}_{(s,1)},\ldots,\hat{S}_{(s,\ell_{\mathcal{H}})}, \hat{Z}_{(z,1)},\ldots,\hat{Z}_{(z,\ell_{\mathcal{H}})}, \hat{R}_{(0,1)},\ldots,\hat{R}_{(0,\ell_{\mathcal{H}})}, \hat{R}_{(1,1)},\ldots,\hat{R}_{(1,\ell_{\mathcal{H}})}) \ .$$

Figure 5: Verifying a proof that the parameters in the issuer's public key were correctly generated.