Direct Anonymous Attestation Explained

Jan Camenisch
IBM Research
jca@zurich.ibm.com

July 25, 2007

1 Introduction

Assume that the user a trusted computing platform communicates with a verifier who wants to be assured that the user indeed uses a platform that can be trusted. trusted hardware module, This problem is called remote attestation and discussed in detail in Chapter **??**. As described there, the problem in the end boils down to the problem that a trusted platform module (TPM) needs to prove that the attestation identity key (AIK) it has generated was indeed generated by an authentic trusted platform module. In principle, the TPM could use its endorsement key (EK) together with its certificate on this key to authenticate the AIK. 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 – thus such a solution does not work as all her transactions would become linkable to each other by the EK.

Another solution to the problem could be 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 [25]. 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 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.

For the TPM V1.2 [26] specification a better solution proposed by Brickell, Camenisch and Chen [4] that was adopted. It is called direct anonymous attestation (DAA) and draws on techniques that have been developed for group signatures [15, 13, 1], identity escrow [20], and credential systems [14, 9]. In fact, the DAA 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, the DAA scheme satisfies an even stronger requirement: when the corrupted software is removed, the privacy/anonymity properties are restored.

As the DAA scheme employs the Camenisch-Lysyanskaya signature scheme [10] 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, DAA uses the Fiat-Shamir heuristic to turn proofs into signatures and thus its security proofs also assume random oracles.

As already mentioned, our setting shares many properties with the one of group signatures [15, 13, 1], identity escrow [20], and credential systems [14, 9] and we employ many techniques [1, 9, 10] 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 subprotocol 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.

This chapter explains the direct anonymous attestation scheme. First, the chapter provides all the building blocks for DAA: the basics for the uses proof of knowledge protocols, the Camenisch-Lysyanskaya (CL) signature

scheme, how one can obtain a CL signature on a secret message, and how to prove knowledge of a CL signature on a secret message. Then, the chapter describes the DAA scheme itself. Finally, a number of applications of the DAA are discussed.

2 Preliminaries

2.1 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 $[-2^\ell + 1, 2^\ell - 1]$.

2.2 Cryptographic Assumptions and Handy Facts

The protocols discussed in this chapter rely on the strong RSA and the decisional Diffie-Hellman assumptions. We state then briefly for completeness.

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 $[0, \rho - 1]$*

The following theorem will turn out to be handy in some of our analyses.

Theorem 1. *[12] Under the strong RSA assumption, given a modulus n (distributed as above), along with random elements $g, h \in (\mathbb{Z}_n^*)^2$, it is hard to compute an element $w \in \mathbb{Z}_n^*$ and integers a, b, c such that*

$$w^c \equiv g^a h^b \pmod{n} \quad \text{and } c \text{ does not divide } a \text{ or } b. \tag{1}$$

Intuitively, computing such a w and integers a, b, and c such that c does not divide a and b seems to require knowledge of the group's order: One computes w as $g^x h^y$ for some x and y and then raises both sides to the power of c. Now if c should not divide a, then one seems to need reducing xc modulo the order of the group. However, this is not possible as the order is not known and so it seems impossible to compute such an a that is not divisible by c.

3 Protocols to Prove Knowledge of and Relations among Discrete Logarithms

In the following 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 [13] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For instance,

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

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, with* $v < \alpha < u$," 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. More precisely, the property of "proof of knowledge" means that there exists a *knowledge extraction* algorithm who can extract the Greek quantities from a successful prover if given rewinding and reset access to the prover (cf. Chapter **??**). Thus, using this notation, a proof protocol can be described by just pointing out its aim while hiding the protocols realization details.

In the following we will first explain how such protocols can be constructed. Unless otherwise stated, we assume a group $G = \langle g \rangle$ of prime order q.

3.1 Schnorr's Identification Scheme

The probably simplest is case is the protocol denoted $PK\{(\alpha) : y = g^\alpha\}$, where $y \in G$, and is depicted in Figure 1. This protocol is also known as Schnorr's identification protocol [24]. As the first step in the protocol, the prover chooses a random integer r_α, computes the *protocol commitment*

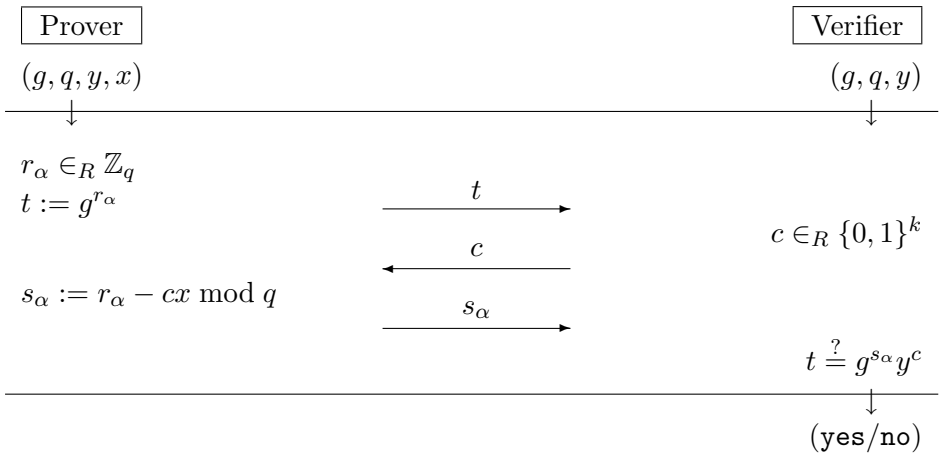

Figure 1: The protocol denoted $PK\{(\alpha) : y = g^\alpha\}$. The prover's input to the protocol is (g, q, y, x) and the verifier's input is (g, q, y). The prover has no output; the verifier's output is either `yes` or `no`, depending on whether or not he accepts the protocols, i.e., whether or not $t = g^{s_\alpha} y^c$ holds.

$t := g^{r_\alpha}$ and sends it to the verifier. The verifier replies with a random *protocol challenge* c. Next, the prover computes the *protocol response* $s := r - cx \bmod q$ and sends it to the verifier. Finally, the verifier accepts the protocol if the *verification equation* $t = g^{s_\alpha} y^c$ holds.

The protocol is a proof of knowledge of the discrete logarithm $\log_g y$ with cheating probability (knowledge error) of 2^{-k}. The protocol is also zero-knowledge against an honest verifier.

To achieve zero-knowledge against any verifier, one needs to choose k logarithmic in the security parameter and repeat the protocols sufficiently many times to make the knowledge error small enough, losing some efficiency by this repetition. Reasonable parameters seem to be $k = 10$ and repeating the protocol 8 times to achieve an overall cheating probability of 2^{-80}. Luckily, one can alternatively use one of the many known constructions to achieve zero-knowledge that retain efficiency, e.g., [16]. We note that this discussion holds for all the protocols we consider in this chapter.

From the protocol just discussed, one can derive the Schnorr signature scheme, denoted $SPK\{(\alpha) : y = g^\alpha\}(m)$, by using the Fiat-Shamir heuristic [17, 23], i.e., by replacing the verifier by a call to a hash function $\mathcal{H}$ and thus computing the challenge as $c = \mathcal{H}(g \| y \| t \| m)$, where $m \in \{0,1\}^*$ is the message that is signed. The signature of m consists of the pair (s, c). The

verification equation of the signature scheme entails computing $\hat{t} := g^s y^c$ and then verifying whether $c = \mathcal{H}(g\|y\|\hat{t}\|m)$ holds. This signature scheme can be shown secure in the so-called random oracle model [3].

3.2 Proving Knowledge of a Representation

One generalization of Schnorr's identification scheme is to use, say, ℓ bases $g_1, \ldots, g_\ell$ with $g_i \in G$. That is, the protocol denoted $PK\{(\alpha_1, \ldots, \alpha_\ell) : y = \prod_{i=1}^{\ell} g_i^{\alpha_i}\}$ is a proof of knowledge of the representation of $y \in G$ w.r.t. the bases g_i. It is constructed as follows. The inputs of the prover and the verifier consist of y, g_1, ..., g_ℓ, the order q of the group, and the system parameter k. The secret input of the prover consists of x_i's such that $y = \prod_{i=1}^{\ell} g_i^{x_i}$. Thus, each x_i corresponds to an α_i. Now, to compute the first message of the protocol, the prover chooses ℓ random integers $r_{\alpha_i} \in \mathbb{Z}_q$, computes $t := \prod_{i=1}^{\ell} g_i^{r_{\alpha_i}}$, and sends t to the verifier. The verifier replies with a c chosen as in the protocol above, i.e., randomly from $\{0,1\}^k$. Next, the prover computes $s_{\alpha_i} := r_{\alpha_i} - cx_i \bmod q$ and sends the resulting $s_{\alpha_1}, \ldots, s_{\alpha_\ell}$ to the verifier. The verifier will accept the protocol if the equation $t = y^c \prod_{i=1}^{\ell} g_i^{s_{\alpha_i}}$ holds. This protocol can be shown to be a proof of knowledge of values α_i such that $y = \prod_{i=1}^{\ell} g_i^{\alpha_i}$ with knowledge error 2^{-k} and to be honest-verifier zero-knowledge.

3.3 Combining Different Proof Protocols

One can combine different instances of the protocol described so far. The simplest combination is a protocol denoted

$$PK\{(\alpha_1, \ldots, \alpha_\ell, \beta_1, \ldots, \beta_{\ell'}) \quad : \quad y \quad = \quad \prod_{i=1}^{\ell} g_i^{\alpha_i} \quad \wedge \quad z \quad = \quad \prod_{i=1}^{\ell'} h_i^{\beta_i}\}$$

with $g_i, h_i, y, z \in G$. It is obtained by running the two protocols

$$PK\{(\alpha_1, \ldots, \alpha_\ell,) : y = \prod_{i=1}^{\ell} g_i^{\alpha_i}\} \quad \text{and} \quad PK\{(\beta_1, \ldots, \beta_{\ell'}) : z = \prod_{i=1}^{\ell'} h_i^{\beta_i}\}$$

in parallel as sub-protocols as follows. First, the prover computes and sends to the verifier the commitment messages of both protocols at the same time. Next, the verifier chooses and sends back a *single* challenge message, i.e., the verifier chooses the same challenge message for both protocols. Finally, the verifier will accept the overall protocol only if the verification equations of both (sub-)protocols hold.

In the same way one constructs a protocol that involves several terms (i.e., representations of several values) by just running the protocol for each term in parallel and by letting the challenge to be the same for each of these protocols. So, for instance, the protocol $PK\{(\alpha, \beta, \gamma) : y = g^\alpha \ \wedge \ z = h^\beta \ \wedge \ w = g^\gamma\}$ is obtained by running the three sub-protocols $PK\{(\alpha) : y = g^\alpha\}$, $PK\{(\beta) : z = h^\beta\}$, and $PK\{(\gamma) : w = g^\gamma\}$ in parallel in as just described.

3.4 Proving Equality of Discrete Logarithms

Another useful combination of the protocols discusses so far is one where, in addition to prove knowledge of discrete logarithms or representations, the prover can show that some discrete logarithms are equal. Such protocols are in principle also obtained from running the basic protocol for each term in parallel. However, now not only are the challenges the same for each of these protocols but also some of random choices of the prover as well as some of the response messages of the protocols needs to be the same. In general, when we compose protocols for arbitrary terms, the prover is to use the same random r_α in all the protocols that involve α in their term. Let us consider the protocol $PK\{(\alpha, \beta) : y = g^\alpha h^\beta \ \wedge \ z = h^\alpha\}$ as an example. The verifier's and the prover's common input to the protocol consists of y, z, g, h, q and the prover's secret input consists of x_α and x_β such that $y := g^{x_\alpha} h^{x_\beta}$ and $z := h^{x_\alpha}$. To compute the commitment message, the prover chooses random r_α and r_β from $\mathbb{Z}_q$ and computes $t_y := g^{r_\alpha} h^{r_\beta}$ and $t_z := h^{r_\alpha}$. Upon receiving the commitment messages t_y and t_z, the verifier replies with a single random challenge message $c \in_R \{0,1\}^k$. Next, the prover computes the response messages as $s_\alpha := r_\alpha + cx_\alpha \bmod q$ and $s_\beta := r_\beta + cx_\beta \bmod q$. Having received s_α and s_β, the verifier will accept the protocol if the two verification equations $t_y = y^c g^{s_\alpha} h^{s_\beta}$ and $t_z = z^c h^{s_\alpha}$ hold.

Let us explain why the verifier should be convinced that $\log_h z$ equals the first element in the representation of y w.r.t. g and h. Using standard rewinding techniques, one can obtain from a successful prover commitment and response messages for different challenge messages but the same commitment messages, i.e., two tuples $(t_y, t_z, c, s_\alpha, s_\beta)$ and $(t'_y, t'_z, c', s'_\alpha, s'_\beta)$ with $(t_y, t_z) = (t'_y, t'_z)$ and $c \neq c'$. From the verification equations one can thus conclude that

$$y^{c-c'} = g^{s'_\alpha - s_\alpha} h^{s'_\beta - s_\beta} \quad \text{and} \quad z^{c-c'} = h^{s'_\alpha - s_\alpha} \ .$$

Now we can set $\hat{x}_\alpha := (s'_\alpha - s_\alpha)(c - c')^{-1} \bmod q$ and $\hat{x}_\beta := (s'_\beta - s_\beta)(c - c')^{-1} \bmod q$ and thus we have

$$y = g^{\hat{x}_\alpha} h^{\hat{x}_\beta} \quad \text{and} \quad z = h^{\hat{x}_\alpha} \ ,$$

i.e., we see that indeed $\log_h z$ equals the first element in the representation of y w.r.t. g and h.

From what we have now seen, we are able to construct protocols that fall into the class denoted

$$PK\{(\alpha_1, \ldots \alpha_{\ell_\alpha}) : y_1 = \prod_{i \in I_1} g_i^{\alpha_{f_1(i)}} \ \wedge \ y_2 = \prod_{i \in I_2} g_i^{\alpha_{f_2(i)}} \ \wedge \ \cdots \ \wedge \ y_{\ell_y} = \prod_{i \in I_{\ell_y}} g_i^{\alpha_{f_{\ell_y}(i)}}\} \ ,$$

where

- ℓ_α, ℓ_g, and ℓ_y are parameters denoting the number of secrets α_i, of bases g_i, and of elements y_i, respectively,
- the y_i's and g_i's can be arbitrary elements of G and do not necessarily be distinct or can be a product of other (given) elements, e.g., $y_3 = y_5 g_2^3 / y_2$,
- the sets I_j define which bases are used for the j-th term and the functions f_j define which secret α_k is used for a particular base in the term.

The protocol is depicted in Figure 2. We note that the protocol has the property that from a successful prover one can extract values $\alpha_1, \ldots, \alpha_{\ell_\alpha}$ such that the equations

$$y_1 = \prod_{i \in I_1} g_i^{\alpha_{f_1(i)}}, \quad y_2 = \prod_{i \in I_2} g_i^{\alpha_{f_2(i)}}, \quad \cdots, \quad y_{\ell_y} = \prod_{i \in I_{\ell_y}} g_i^{\alpha_{f_{\ell_y}(i)}}$$

all hold. That is, it is an honest-verifier zero-knowledge proof of knowledge of the values $\alpha_1, \ldots, \alpha_{\ell_\alpha}$ with knowledge error 2^{-k}.

Let us finally consider some example instances of this general protocol. The protocol denoted $PK\{(\alpha, \beta) : y_1 = g^\alpha \wedge y_2 = g^\beta \wedge y_3 = y_1^\beta\}$ proves that $\log_g y_3$ is the product of $\log_g y_1$ and $\log_g y_2$, the protocol denoted $PK\{(\alpha) : y_1 = g^\alpha \wedge y_2 = y_1^\alpha\}$ proves that $\log_g y_2 = (\log_g y_1)^2$, and the protocol denoted $PK\{(\alpha, \beta) : g = y^\alpha h^\beta\}$ proves that the first element of the representation of y that the prover knows w.r.t. g and h is non-zero, provided that the prover is not privy to $log_g h$ (in case the prover knows $log_g h$, he is able to compute q different representations of y w.r.t. g and h, otherwise he can only know one).

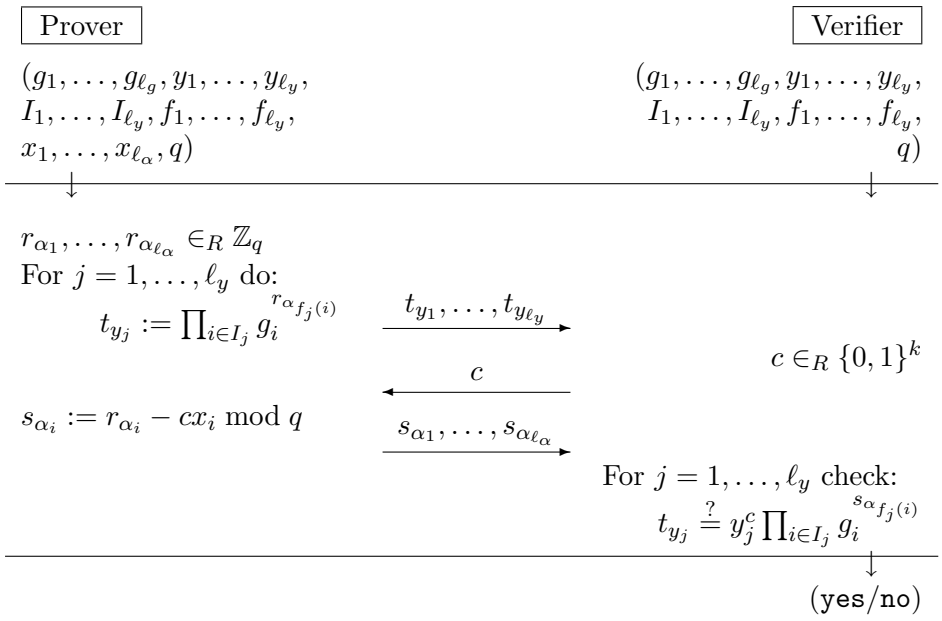

Figure 2: The protocol denoted $PK\{(\alpha_1, \ldots \alpha_{\ell_\alpha}) : y_1 = \prod_{i \in I_1} g_i^{\alpha_{f_1(i)}} \wedge y_2 = \prod_{i \in I_2} g_i^{\alpha_{f_2(i)}} \wedge \cdots \wedge y_{\ell_y} = \prod_{i \in I_{\ell_y}} g_i^{\alpha_{f_{\ell_y}(i)}}\}$.

3.5 The Schnorr Protocol Modulo a Composite.

So far we have considered proof protocols for a group of prime order where the order is known to both the prover and the verifier. However, often one would like to use these kind of protocols in groups where the order is not known to all parties as is for instance the case in RSA-groups. RSA groups are subgroups of $\mathbb{Z}_n^*$, where n is the product of two primes. Thus, if one does not know the factorization of n, one does in general not know the order of a subgroup generated by a random element of $\mathbb{Z}_n^*$.

In particular, let n be the product of two safe primes, i.e., primes p and q such that $p' = (p-1)/2$ and $q' = (q-1)/2$ are also primes. Let g be a generator of the quadratic residues modulo n (so, g will have order $p'q'$). In this case, knowing the order $p'q'$ of g is equivalent to knowing the factorization of n. Let $y = g^x$ with $x \in \{0,1\}^\ell$ for some ℓ. Now consider the protocol denoted $PK\{(\alpha) : y = g^\alpha \pmod n\}$, where at least the prover is assumed not to be privy to the factorization of n. The prover and verifier's common inputs are (y, g, n, ℓ_n) and the prover's additional

input is $x = \log_g y$, where $\ell_n = \lceil \log_2 n \rceil$ (i.e., the length of n in bits). We may assume that $x \in [0, n]$. The protocol is as follows.

1. The prover chooses uniformly at random $r_\alpha \in_R \{0,1\}^{\ell_n+\ell_c+\ell_\varnothing}$, computes $t_y := g^r \bmod n$, and sends t to the verifier.
2. The verifier chooses a random $c \in_R \{0,1\}^{\ell_c}$ and sends that to the prover.
3. The prover replies with $s_\alpha := r_\alpha - xc$.
4. The verifier checks whether $t_y \stackrel{?}{\equiv} y^c g^{s_\alpha} \pmod n$ holds.

The difference to the protocol in the case the order of the group $\langle g \rangle$ is known is that here, as she does not know the order of the group, the prover can no longer choose r_α randomly from the integers modulo this order and can no longer reduce s_α modulo this order. So, the prover needs to choose r_α some how differently such that nevertheless s_α and t_y do not reveal information about x, i.e., such that the protocol remains zero-knowledge. Thus if $x \in [0, n]$ and the prover chooses $r_\alpha \in_R \{0,1\}^{\ell_n+\ell_c+\ell_\varnothing}$, then, for any $c \in \{0,1\}^{\ell_c}$ and sufficiently large $\ell_\varnothing$ (e.g., 80), the value $s_\alpha := r_\alpha - xc$ is distributed statistically close to the uniform distribution over $\{0,1\}^{\ell_n+\ell_c+\ell_\varnothing}$. Also, the value $t_y := g^r \bmod n$ is distributed statistically close to uniform over $\langle g \rangle$. Therefore, provided that $y \in \langle g \rangle$, the protocol is statistical honest-verifier zero-knowledge for sufficiently large $\ell_\varnothing$ (e.g., $\ell_\varnothing = 80$).

Now, this protocol is only a proof of knowledge of $\log_g y$ if ℓ_c equals 1 and if repeated sufficiently many, say k, times. Let us investigate this. Assume we are given a prover that can successfully run the protocol for given y, g, and n. By standard rewinding techniques, one can extract two triples (t, c, s) and (t', c', s') from the prover such that $t = t'$, $c \neq c'$, and $t \equiv y^c g^s \equiv y^{c'} g^{s'} \pmod n$ holds. W.l.o.g. we may assume that $c' > c$. From the last equation we can derive that $y^{c'-c} \equiv g^{s-s'} \pmod n$.

If $\ell_c = 1$, we must have $c' = 1$ and $c = 0$ and hence $y \equiv g^{s-s'} \pmod n$, i.e., $s - s'$ is a discrete logarithm of y to the base g and hence the protocol is indeed a proof of knowledge.

If $\ell_c > 1$, we are stuck with the equation $y^{c'-c} \equiv g^{s-s'} \pmod n$, i.e., we seem to need to compute a $(c' - c)$-th root of $g^{s-s'}$ modulo n which is assumed to be hard without knowledge of g's order. Unfortunately, Theorem 1 provides a way out. That is, under the strong RSA assumption, we will have that $(c' - c) | (s - s')$. Let u be such that $(c' - c)u = (s - s')$. Then $y \equiv b g^u \pmod n$ for some b such that $b^{c'-c} \equiv 1 \pmod n$. Assuming that

$2_c^\ell < \min(p', q')$, it must be that $b = \pm 1$ or $\gcd(b \pm 1, n)$ splits n (which again would counter the strong RSA assumption). Of course, if both y and g are quadratic residues, then $y \equiv g^u \pmod{n}$.

The reader might now think that the protocol is therefore a proof of knowledge under the strong RSA assumption for $\ell_c > 1$. Unfortunately this is not the case. Let us expand. The definition of a proof of knowledge (cf. Chapter **??**), requires that the inputs n, g, and y be fixed, i.e., that the knowledge extractor works for any prover that is successful for a given input n, g, and y. However, the above argument only works for n chosen at random. For instance, it does not work if the prover knew the factorization of n as he then could compute c, c', s, and s' such that $(c' - c) \nmid (s - s')$ (for instance by adding a multiple of the order of g to s). Now, if n is fixed, there always exists a prover who has the factorization encoded into herself and thus she could successfully run the protocol but the knowledge extractor cannot extract a witness. In order words, the protocol only has the proof of knowledge property for random n which contradicts the requirement of a proof of knowledge that the witness can be extracted for any given n. Nevertheless, the protocol is still useful as a building block, i.e., one only need to take into account the probability spaces of n, g, and y. That is, one has to consider the overall system of which the protocol is part and cannot just consider the protocol by itself as one could if it was a true proof of knowledge. Despite of all of this, we denote this protocol also as $PK\{(\alpha) : y \equiv \pm g^\alpha \pmod{n}\}$ or $PK\{(\alpha) : y \equiv g^\alpha \pmod{n}\}$, depending on whether the verifier is assured that y is a quadratic residue or not.

3.6 Proving that a Secret Lies in a Given Interval

One property of the protocol just described that is handy in many cases is the fact that the prover cannot reduce the response messages modulo the order of the group is to argue about the bit-length of the secret known to the prover. Let $x \in \pm\{0,1\}^{\ell_x}$ for some ℓ_x and let $y := g^x \bmod n$, with g and n as before. Now consider the following modification of the protocol (the inputs to the prover and the verifier remain unchanged except that both parties are further given ℓ_x).

1. The prover chooses uniform at random $r_\alpha \in_R \{0,1\}^{\ell_x + \ell_c + \ell_\varnothing}$, computes $t_y := g^r \bmod n$, and sends t to the verifier.

2. The verifier chooses a random $c \in_R \{0,1\}^{\ell_c}$ and sends that to the prover.

3. The prover replies with $s_\alpha := r_\alpha - xc$.

4. The verifier checks whether

$$t_y \stackrel{?}{\equiv} y^c g^{s_\alpha} \pmod{n} \quad \text{and} \quad s_\alpha \stackrel{?}{\in} \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+1}$$

hold.

The modification is that the prover chooses r_α from a different interval and that the verifier also checks that s_α takes at most $\ell_x + \ell_c + \ell_\varnothing + 1$ bits.

The analysis of this protocols is of course almost identical to the original one, except that we are now considering the bit-lengths of s_α and r_α. First, it is not difficult to see that the protocol remains statistical honest-verifier zero-knowledge. Above we have argued that under the strong RSA assumption, one can extract a value u from a successful prover such that $y \equiv \pm g^u \pmod{n}$, where with $u = (s-s')/(c'-c)$. Now because $(c'-c)$ divides $(s-s')$ and as $(s-s') \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}$, we must have $u \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}$. In other words, the discrete logarithm known to the prover has at most $\ell_x + \ell_c + \ell_\varnothing + 2$ bits (neglecting its sign). Note that this length is independent of the length of the modulus n. Also note that in fact the length of the prover's secret must be shorter, (only about ℓ_x bits) for the prover to be able to run the protocol successfully with high probability; so the protocol is not an argument of the exact length of the secret. However, in many cases this is good enough. We denote this modified protocol as $PK\{(\alpha) : y \equiv g^\alpha \pmod{n} \quad \wedge \quad \alpha \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}\}$.

The protocol can be also be used to prove that the secret known to the prover lies in any interval, say, in $[a,b]$. To this end note that $[a,b] = [\frac{a+b}{2} - \frac{b-a}{2}, \frac{a+b}{2} + \frac{b-2}{2}]$. Thus the protocol denoted

$$PK\{(\alpha) : \frac{y}{g^{(a+b)/2}} \equiv g^\alpha \pmod{n} \quad \wedge \quad \alpha \in [-\frac{b-a}{2}, \frac{b-a}{2}]\}$$

is an argument that the prover knows a value x such that $x \in [a,b]$ and $y \equiv g^x \pmod{n}$. As before, the actual value x given as input to the prover must lie in a smaller interval, namely in the interval $[\frac{a+b}{2} - \frac{b-a}{2\cdot 2^{\ell_c+\ell_\varnothing+2}}, \frac{a+b}{2} + \frac{a+b}{2\cdot 2^{\ell_c+\ell_\varnothing+2}}]$.

It is straightforward to extent and generalize the protocols just discussed in the same way as the protocols we considered for groups of known order. Moreover, the protocols over groups of known order and those over groups of unknown order can be combined. Let us consider a simple combination as an example; the constructions of general combinations is left as an exercise to the reader.

Assume a group $G = \langle g \rangle$ of order a prime q, and let n be an RSA modulus as above, h_1 be an element from $\mathbb{Z}_n^*$, h_2 be an element from $\langle h_1 \rangle$, and $y = g^x$ with $x \in \{0,1\}^{\ell_x}$ for some integer $\ell_x < (\log_2 q) - 1 - (\ell_c + \ell_\varnothing + 2)$. Finally, assume that the prover is not privy to n's factorization. Now consider the following protocol. The common input to the prover and the verifier consists of q, g, y, n, h_1, h_2, and a and the secret input to the prover consists of x.

1. First, the prover chooses a random $r \in_R [0, n2^{\ell_\varnothing}]$, computes $z := h_1^x h_2^r \bmod n$, and sends z to the verifier.

2. Next, the prover and the verifier run the protocol denoted:

$$PK\{(\alpha,\beta) : y = g^\alpha \ \wedge \ z \equiv h_1^\alpha h_2^\beta \pmod{n} \ \wedge \ \alpha \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}\},$$

   i.e., they perform the following steps.

   (a) The prover chooses a random $r_\alpha \in [-a2^{\ell_c+\ell_\varnothing}, a2^{\ell_c+\ell_\varnothing}]$ and a random $r_\beta \in \{0,1\}^{\ell_n+\ell_c+2\ell_\varnothing}]$, computes $t_y := g^{r_\alpha}$ and $t_z := h_1^{r_\alpha} h_2^{r_\beta} \bmod n$ and sends t_y and t_z to the verifier.

   (b) The verifier chooses a random $c \in_R \{0,1\}^{\ell_c}$ and sends that to the prover.

   (c) The prover replies with $s_\alpha := r_\alpha - xc$ and $s_\beta := r_\beta - rc$.

   (d) The verifier checks whether

$$t_y \stackrel{?}{=} y^c g^{s_\alpha} \quad , \qquad t_z \stackrel{?}{\equiv} z^c h_1^{s_\alpha} h_2^{s_\beta} \pmod{n} \quad , \text{ and}$$
$$s_\alpha \stackrel{?}{\in} \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+1}$$

   hold.

Let us analyze this protocol. From our considerations above we know that one can extract from a successful prover values $(t_y, t_z, c, s_\alpha, s_\beta)$ and $(t'_y, t'_z, c', s'_\alpha, s'_\beta)$ with $(t_y, t_z) = (t'_y, t'_z)$ and $c \neq c'$. From the verification equations we have that

$$z^{c-c'} \equiv h_1^{s'_\alpha - s_\alpha} h_2^{s'_\beta - s_\beta} \pmod{n} \ , \qquad y^{c-c'} = g^{s'_\alpha - s_\alpha} \ , \quad \text{and}$$
$$(s'_\alpha - s_\alpha) \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2} \ .$$

From the first of these equations we can conclude that under the Strong RSA assumption $(c - c')$ divides $(s'_\alpha - s_\alpha)$, i.e., there exists some u such that $(s'_\alpha - s_\alpha) = u(c - c')$. Thus, we can rewrite the second equation as

$y^{c-c'} = g^{u(c-c')}$. Now, as $y \in \langle g \rangle$ (which we can test as g has prime order q), we must have $y = g^u$. From the third of the equations we can further derive that $u \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}$. Thus the verifier is assured that the prover knows $\log_g y$ which lies in $\pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}$, i.e., the protocol is a proof that a discrete logarithm in a group of *known order* lies in some interval. Of course this makes sense only if the group's order is larger than $2^{\ell_x+\ell_c+\ell_\varnothing+3}$.

As the basic protocol $PK\{(\alpha) : y \equiv \pm g^\alpha \pmod{n}\}$ in group of unknown order, the protocol just described in not a proof of knowledge either as its analysis depends on the strong RSA assumption and thus is correct only if the prover can execute the protocol for a random n. As the goal of the protocol was to prove that $\log_g y$ lies in some interval, we do not need to fix n but could have the verifier generate n and send it to the prover before running the protocol. The protocol augmented like this would indeed be a true proof of knowledge. For the protocol to be zero-knowledge, the prover needs to be ensured that $h_2 \in \langle h_1 \rangle$ as otherwise z could leak information about x. Unfortunately, the only way known to prove that $h_2 \in \langle h_1 \rangle$ holds is not very efficient, i.e., is to have the verifier run the protocol $PK\{(\alpha) : h_2 \equiv \pm h_1^\alpha \pmod{n}\}$ many times with binary challenges (cf. above). However, in some cases this proof can be delegated to a set-up phase and thus needs to be done only once and for all, or sometimes n, h_1, and h_2 can be provided by a trusted third party.

The protocol just discussed can be extended to three (or more) different groups, i.e., groups $G_1 = \langle g_1 \rangle$ and $G_2 = \langle g_2 \rangle$ of known order q_1 and q_2 together with a RSA sub-group of unknown order as to show that two discrete logarithms in G_1 and G_2 are the same. I.e., let $y_1 = g_1^x$ and $y_2 = g_2^x$ with $x \in \pm\{0,1\}^{\ell_x}$ and $\ell_x < (\log_2 \min\{q_1, q_2\}) - 1 - (\ell_c + \ell_\varnothing + 2)$, and $z = h_1^x h_2^r \pmod{n}$ for some random r. Then, the protocol

$$PK\{(\alpha,\beta) : y_1 = g_1^\alpha \quad \wedge \quad y_2 = g_2^\alpha \quad \wedge \quad z \equiv h_1^\alpha h_2^\beta \pmod{n} \quad \wedge \quad \alpha \in \pm\{0,1\}^{\ell_x+\ell_c+\ell_\varnothing+2}\} \ ,$$

will convince a verifier that $\log_{g_1} y_1 = \log_{g_2} y_1$ where we define $\log_g y$ to be the integer x for which $y = g^x$ and that closed to 0, i.e., $\log_g y$ lies in $[-q/2, q/2]$. Of course, the protocol can also be generalized to representations, i.e., to a protocol showing that an element of a representation lies in a certain interval.

3.7 The Camenisch-Lysyanskaya Signature Scheme

Recall that the idea underlying the direct anonymous attestation scheme is that a TPM gets from a attester a signature on a secret message and then

later proves knowledge of such a signature on a secret message. While it is known that the latter statement can always be proven, such a proof would in general not be very efficient.

Luckily, Camenisch-Lysyanskaya provide a signature scheme [11, 22]. that, unlike most signature schemes, is particularly suited for our purposes: it allows one to apply the proof efficient discrete logarithm proof protocols just discussed to prove knowledge of a signature and to retrieve signatures on secret messages [11, 22].

We recall this signature scheme (the CL signature scheme) and the related protocols here.

Key generation. On input ℓ_n, choose an ℓ_n-bit RSA modulus n such that $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$, where p, q, p', and q' are primes. 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.

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

$$A = \left(\frac{Z}{R_0^{m_0} \ldots R_{L-1}^{m_{L-1}} S^v} \right)^{1/e} \mod n \ .$$

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

$$Z \equiv A^e R_0^{m_0} \ldots R_{L-1}^{m_{L-1}} S^v \pmod{n} \ , \qquad m_i \in \pm\{0,1\}^{\ell_m} \ , \quad \text{and}$$
$$2^{\ell_e} > e > 2^{\ell_e - 1}$$

all holds.

Theorem 2 ([11])**.** *The signature scheme is secure against adaptive chosen message attacks [19] under the strong RSA assumption.*

The original scheme considered messages in the interval $[0, 2^{\ell_m} - 1]$. Here, however, we allow messages to be 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 $\ell_e > \ell_m + 1$.

3.8 Obtaining a Camenisch-Lysyanskaya Signature on a Secret Message

Camenisch and Lysyanskaya also provided a protocol that allows a user to obtain a signature on messages that the signer does not learn. The protocol we present here is a somewhat more efficient version than the one by Camenisch and Lysyanskaya [11] as it incorporates improvements presented by Brickell, Camenisch, and Chen [4] and by Camenisch and Groth [8].

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. Section 4.4 for the latter).

A signature on messages $m_0, \ldots, m_{L-1} \pm \{0,1\}^{\ell'_m}$, such that only the messages $m_{L'}, \ldots m_{L-1}$ are known to the signer $\mathcal{S}$, can be obtained by a signature receiver $\mathcal{R}$ as follows. The parameter ℓ'_m should be at most $\ell_m - \ell_\varnothing - \ell_\mathcal{H} - 2$. The reason for this is that the protocols that the user (signature receiver) can apply to convince the signer that the messages lie in $\pm\{0,1\}^{\ell_m}$ has a fudge factor of $2^{\ell_\varnothing + \ell_\mathcal{H} + 2}$ as we have explained (cf. 3.5).

The protocols is as follows. The input of the signature receiver $\mathcal{R}$ consists of the signer $\mathcal{S}$'s public key and all its messages. The input of $\mathcal{S}$ consists of its secret and public keys and of the messages $m_{L'}, \ldots m_{L-1}$. The parties execute the following steps.

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

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

$$PK\{(\mu_0, \ldots, \mu_{L'-1}, \nu') : \; U \equiv \pm S^{\nu'} \prod_{i=0}^{L'-1} R_i^{\mu_i} \pmod n \; \wedge \; \mu_0, \ldots, \mu_{L'-1} \in \pm\{0,1\}^{\ell_m}\} \; .$$

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

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

   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

$$PK\{(\delta) : \quad A \equiv \pm\Big(\frac{Z}{U S^{v''} \prod_{i=L'}^{L-1} R_i^{m_i}}\Big)^{\delta} \pmod n\}$$

with $\mathcal{R}$ as verifier.

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

Note that $\mathcal{R}$ is not privy to 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., e.g., [4]).

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 reveal an e-th root of some value $\mathcal{R}$ concocted. Finally, the proof in Step 4 will convince $\mathcal{R}$ that $\mathcal{S}$ computed 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 as we will see in the next section. Let us argue why it is true. From the properties of the PK protocol, we know that one can extract values c and d from a successful prover such that

$$A^c \equiv \big(\frac{Z}{US^{v''}\prod_{i=L'}^{L-1} R_i^{m_i}}\big)^d \equiv \big(\frac{Z}{S^{v}\prod_{i=0}^{L-1} R_i^{m_i}}\big)^d \pmod{n} \ ,$$

where c is the difference of two challenge messages and hence must lies in $\in \{0,1\}^{\ell_{\mathcal{H}}}$. Furthermore, we know that

$$Z \equiv A^e S^v \prod_{i=0}^{L-1} R_i^{m_i} \pmod{n}$$

holds and that e is a prime. Furthermore, there must exits integers a_1 and a_2 such that $ea_1 + ca_2 = 1$ as $c \nmid e$ (provided that c is smaller than e which can easily be ensured and is usually satisfied because of the requirement that one be able to prove that e is sufficiently large). Thus we have

$$A \equiv A^{a_1 e + c a_2} \equiv \big(\frac{Z}{S^{v''}\prod_{i=0}^{L-1} R_i^{m_i}}\big)^{a_1 + a_2 d} \pmod{n} \tag{2}$$

and therefore, as the signer has during the key set-up also ensured us that $Z, R \in \langle S \rangle$, the element A must lie in the group generated by S.

Security Considerations. Using the protocol just described to issue signature might render the signature scheme insecure as the signer does no longer learn the signature he signed. However, in the security proof the Camenisch-Lysyanskaya signature scheme the messages need to be known

to reduce the security of the signature scheme to the strong RSA assumption. Therefore, when proving security of the signature schemes together with this protocol to issue signatures on hidden messages, one needs to extract the hidden messages from the signature receiver. The only known means to achieve this with this protocol is via rewinding. Such rewinding, however, can only be successfully done if different instances of the protocol are run sequentially. That is, the signature scheme only retains security of the signer does not run the protocol concurrently with many signature receivers. If the signer needs to be able to run such an issuing protocol concurrently, then the protocol needs to be modified such that (in the security proof) one is able to extract the secret messages without rewinding. One means to achieve this is to use verifiable encryption as follows (cf. [12]). Assume that a public key of a suitable encryption scheme is made available by an trusted third party such that the corresponding secret key is not known. Then, when a signature receiver want to obtain a signature on a hidden message, we require that she also encrypt the message under that public key and prove that the message encrypted is the one contained in the value U. This can efficiently be achieved with the Camenisch-Shoup [12] encryption scheme. Now, the idea is that when proving security, the reduction algorithm controls the third party, thus will know the secret key, and is able to get hold on the secret messages just by decrypting. As this requires no longer any rewinding, the signing protocol can be run concurrently with many signature receivers. An alternative method is due to Fischlin [18] who recently presented a transformation for turning any standard proof of knowledge (or Σ-protocol) into a non-interactive proof in the random oracle model that supports an online extractor (i.e., requires no rewinding for extraction).

3.9 Proving Possession of a Signature

Another protocol that we will need is one to convince a verifier of ones possession of a signature on some messages. Again, this protocol is based on one by Camenisch and Lysyanskaya [11] but incorporates improvements presented by Brickell, Camenisch, and Chen [4] and by Camenisch and Groth [8].

We demonstrate the protocol here for the case where the messages m_i for $i \in I_r$ are revealed to the verifier, the messages m_i for $i \in I_c$ are hidden from the verifier but of which she receives a commitment, and finally no information whatsoever about the messages m_i for $i \in I_h$ is revealed to the verifier.

Let us discuss the ideas underlying the protocol before we give its details. To this end, consider a signature (A, e, v) on a single message m. The

signature is valid if we have that

$$Z \equiv R_0^m S^v A^e \pmod{n} \ , \quad 2^{\ell_e - 1} < e < 2^{\ell_e} \ , \quad \text{and} \quad m \in \pm\{0,1\}^{\ell_m} \ .$$

Of course we want to use the protocols described in Section 3.5. Now, if A was a public value, then we could use these protocols to argue knowledge of a representation of Z w.r.t. R_0, S, and A and that the first and third elements of this representation lie in the required intervals. Making A public, however, would leak information about the signature! Taking a closer look at Camenisch-Lysyanskaya signatures reveals that one can, however, randomize A: Given a signature (A, e, v), the tuple $(A' := AS^{-r} \bmod n, e, v' := v + er)$, is also a valid signature on the message m for any r, as can easily be verified [8]. Now, provided that $A \in \langle S \rangle$ and that r is chosen uniformly at random from $\{0,1\}^{\ell_n + \ell_\varnothing}$, the values A' is distributed statistically close to uniform over $\mathbb{Z}_n^*$ and hence can be revealed without leaking information about the signature (A, e, v). While the second requirement is easy to guarantee by selecting r correctly, it is in general much harder to check that $A \in \langle S \rangle$ holds as the order of S is not known. Therefore, we needed the signer to prove this fact in the protocol for issuing a signature described in the previous section.

Assuming this, to convince a verifier that one possesses a signature on some message by the signer, one can randomize that signature as just shown, send A' to the verifier and then run the protocol

$$PK\{(\varepsilon, \nu', \mu) : \quad Z \equiv \pm R_0^\mu A'^\varepsilon S^{\nu'} \pmod{n} \ \wedge \ \mu \in \pm\{0,1\}^{\ell_m} \ \wedge \ \varepsilon \in [2^{\ell_e - 1} + 1, 2^{\ell_e} - 1]\}$$

with the verifier. However, as we have seen before, the prover can run successfully a protocol proving that $\mu \in \pm\{0,1\}^{\ell_m}$ and $\varepsilon \in [2^{\ell_e - 1} + 1, 2^{\ell_e} - 1]$ holds only if the corresponding secrets m and e known to the prover actually lie in smaller intervals. We therefore need to require that the message m lie in $\{0,1\}^{\ell'_m}$, as for the protocol in the previous paragraph. Now to enable the prover to show that $\varepsilon \in [2^{\ell_e - 1} + 1, 2^{\ell_e} - 1]$, the signer should choose e from $[2^{\ell_e - 1} - 2^{\ell'_e} + 1, 2^{\ell_e - 1} + 2^{\ell'_e} - 1]$ with $\ell'_e < \ell_e - \ell_\varnothing - \ell_{\mathcal{H}} - 3$. This will allow the prover to show that $e - 2^{\ell_e - 1}$ lies in $\pm\{0,1\}^{\ell_\varnothing + \ell_{\mathcal{H}} + 2}$ which implies $e \in [2^{\ell_e - 1} + 1, 2^{\ell_e} - 1]$. So, we get the protocol

$$PK\{(\varepsilon, \nu', \mu) : \quad Z A'^{-2^{\ell_e - 1}} \equiv \pm R_0^\mu A'^\varepsilon S^{\nu'} \pmod{n} \ \wedge \ \mu \in \pm\{0,1\}^{\ell_m} \ \wedge \ \varepsilon \in \pm\{0,1\}^{\ell'_e + \ell_\varnothing + \ell_{\mathcal{H}} + 2}\} \ .$$

Similarly to the protocol to obtain a signature, we now generalize the protocol just discussed to one where the user $\mathcal{R}$ possesses a signatures on several messages and wants to convince a verifier that she indeed does so while revealing some of the messages and keeping others secret. More concretely, assume that the user $\mathcal{R}$ possesses a signature (A, e, v) on messages $m_1, \ldots, m_L$. Let I_h and I_c be sets of indices defining the messages $\{m_i | i \in I_h\}$ the user wants to hide and the messages $\{m_i | i \in I_c\}$ to which she want to send to verifier commitments. All other messages, the user is ready to reveal to the verifier. This leads to the following protocol, that requires as additional input parameters for a commitment scheme: Let $G = \langle g_0 \rangle$ be a group of prime order $q > 2^{\ell_m}$ and let $g_1, \ldots, g_L$ be elements from G.

1. $\mathcal{R}$ chooses $r' \in \{0,1\}^{\ell_n + \ell_\varnothing}$, computes $C := g_0^{r'} \prod_{i \in I_c} g_i^{m_i}$, and sends C to $\mathcal{V}$ and $\{m_i \mid i \notin I_c \cup I_h\}$.

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

$$PK\{(\varepsilon, \tilde{\nu}, \rho', \{\mu_i | i \in I_c \cup I_h\}) :$$

$$Z A'^{-2^{\ell_e - 1}} \prod_{i \notin I_c \cup I_h} R_i^{-m_i} \equiv \pm A'^{\varepsilon} S^{\tilde{\nu}} \prod_{i \in I_c \cup I_h} R_i^{\mu_i} \pmod{n} \wedge$$

$$C \equiv g_0^{\rho'} \prod_{i \in I_c} g_i^{\mu_i} \ \wedge \ \mu_i \in \{0,1\}^{\ell_m + \ell_\varnothing + \ell_{\mathcal{H}} + 2} \ (i \in I_c \cup I_h) \wedge$$

$$\varepsilon \in \pm\{0,1\}^{\ell'_e + \ell_\varnothing + \ell_{\mathcal{H}} + 1}\}$$

   with $\mathcal{V}$ as verifier.

4. The verifier also checks that $m_i \in \pm\{0,1\}^{\ell_m}$ for $i \notin I_c \cup I_h$.

Having the user (prover) to reveal only some of the messages and to send the verifier commitments to other messages will allow one to use this protocol as a sub-protocol. The messages can for instance be used to encode attributes into a credentials. Examples of such attributes include which kind of resource the user is allowed 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 has not expired yet, or, if required by the application, the user could also provide the verifier with an verifiable encryption [7, 12] of an attribute under a trusted third party's public key. For the latter, the user would first provide the verifier with a commitment of that attribute, prove that the commitment indeed contains the attribute found in the certificate, and the will use a verifiable encryption protocols to verifiably encrypt the committed value. In case the attribute is the user's identity, such a verifiable encryption could allow the trusted third party to revoke the user's anonymity, e.g., in case she misuses the anonymous access to the resource.

3.10 Guaranteeing Privacy to the User

In both protocols just described we have made some assumptions about the issuer's public key in order to guarantee privacy to the user. That is, in the protocol to issue a CL signature on hidden messages, we have assumed that $R_0, \ldots, R_{L-1} \in \langle S \rangle$ as requirement that U is a statistically hiding commitments. In the protocol, to prove knowledge of a CL signature we needed to ensure that $A \in \langle S \rangle$ which in turn required that $Z \in \langle S \rangle$. Let us therefore discuss how the signer can prove that his public key meets these requirements.

While for some groups (e.g., suitable subgroups of $\mathbb{Z}_p^*$ where p is a prime)these requirements can be easily tested by raising the elements by raising the elements to this orders and checking whether the result is the identity element. In case the order should not be revealed, the only known way is to have the issuer prove knowledge of $\log_S Z$. As we have discussed above, such a proof does only work with binary challenges and therefore needs to be repeated a sufficient number of times. The issuer would need to do that for Z as well as for all values R_i. At at first glance this seems somewhat inefficient. However, first of all, using the non-interactive versions for the proof protocols, the issuer would need to generate this proof only once. Moreover, all these protocols use the same base S and hence one can employ a number of techniques to speed-up the generation and the verification of this proofs such as precomputation and also so-called batch verification techniques [2]. Also, one could delegate the verification of the public key's correctness to the authority (CA) certifying the issuer's public key or other suitable trusted parties.

4 The Direct Anonymous Attestation Scheme (DAA)

4.1 Informal Model of Direct Anonymous Attestation

This section informally discusses the functional and security requirements for direct anonymous attestation. For a formal model we refer the reader to [4].

The participants of the scheme are a number of platforms, each consisting of a host and a TPM, an attester, and a number of verifiers. Let us clarify our use of the terms platform, host, and TPM. Each platform contains a TPM. By host we designate the platform excluding the TPM. We assume that each TPM possesses a so-called endorsement key pair and certificate, i.e., an RSA key pair the public key of which has being certified by the TPM's manufacturer.

The scheme consists of three procedures: the *DAA key generation* run by the attester; the *DAA-Join* protocol executed between a host, a TPM, and an attester; and the *DAA-Sign* protocols run between a host, a TPM, and a verifier.

The DAA key generation generates the public parameters of the scheme and the attester's DAA secret key.

The DAA-Join protocol allows a platform to obtain attestation provided it has access to (contains) a certified (endorsed) TPM. The common inputs for this protocol consist of the attester's public key and the TPM's certified endorsement key. The attester has as additional input it's DAA secret key and the TPM it's endorsement secret key. As a result of the protocol the host and the TPM together obtain attestation.

The DAA-Sign protocol allows a platform that has obtained attestation to convince a verifier of this fact. The parties inputs for this protocol are the DAA public parameters and, if the verifier decides so, the verifier's base-name. The TPM and the host have as secret inputs their respective outputs of the DAA-Join protocol.

The security requirements are as follows:

Unforgeability. A TPM and host can successfully run the DAA-Sign protocol with a verifier only if they had at some earlier time successfully run the DAA-Join protocol which in turn is only possible if the TPM possesses a certified endorsement key.

Anonymity. The information the verifier obtains from the DAA-Sign protocol is not sufficient to link the transaction to an instance of the DAA-Join or an other instance of the DAA-Sign protocol. The latter

needs to hold only if the DAA-Sign protocol were run with different base-names.

Rogue Tagging. Given all DAA related (secret) information of a (rogue) TPM, the verifier can tell whether or not he runs the DAA-Sign protocol with a party using this rogue information.

4.2 High-Level Description.

The basic idea underlying the direct anonymous attestation scheme is similar to the one of the Camenisch-Lysyanskaya anonymous credential system [9, 11, 22]: A trusted hardware module (TPM) chooses a secret "message" f, obtains a Camenisch-Lysyanskaya (CL) signature (aka attestation) on it from the attester (issuer) via a secure two-party protocol, and then can convince a verifier that it got attestation anonymously by proving knowledge of an attestation (i.e., of a signature on a secret message). To allow the verifier to recognize rogue TPMs, the 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 message 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, for efficiency reasons, we split the TPM's secret f into two ℓ_f-bit messages and denote these (secret) messages by f_0 and f_1 (instead of m_0 and m_1 as done in the protocols presented in the preceding section). This split allows the issuer to choose smaller primes e as their size depends on the size of the messages that get signed - this will save the issuer/attester considerable computations as the generation of a prime takes time linear in the length of the prime. Now, the two-party protocol to sign secret messages is as follows (cf. [11]). 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 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 and has verified that U and N_I where indeed generated

and set by a valid TPM, 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 same 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 secrets contained 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 base name 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 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 base name then, whenever the same base name $\mathtt{bsn}_V$ 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 base name 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 base name 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.

From this high-level description and the preceding sections, it should be clear how the DAA-Join and DAA-Sign protocols work. However, the design goal of the protocol was to have the TPM to perform as few operations of the signature receiver but to instead have the platform doing them. Indeed, investigating the trust relationships and the protocols, it turns out that some operations that would be carried out by the TPM if one implemented the scheme as just discussed can be outsourced to the host in which the TPM is embedded: it is sufficient if the TPM performs only the operations that are critical to security, i.e., to ensure that attestation cannot be proved without the use of the TPM to which the attestation was issued. All the other operations of the signature receiver, in particular the ones that ensure privacy, can be performed by the host (the host could always destroy the privacy of the protocol anyway, e.g., by reveal a TPM's identity). This requires of course some modification of the previous section's protocols to obtain a signature and to prove knowledge of a signature.

4.3 System 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 reasonable examples for these parameters). We require that: $\ell_e > \ell_\varnothing + \ell_{\mathcal{H}} + \max\{\ell_f + 4\ , \ell'_e + 2\}$, $\ell_v > \ell_n + \ell_\varnothing + \ell_{\mathcal{H}} + \max\{\ell_f + \ell_r + 3\ ,\ \ell_\varnothing + 2\}$, 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 [21]).

The scheme also requires $H_\Gamma(\cdot)$ and $H(\cdot)$, two collision resistant hash functions $H_\Gamma(\cdot) : \{0,1\}^* \to \{0,1\}^{\ell_\Gamma + \ell_\varnothing}$ and $H(\cdot) : \{0,1\}^* \to \{0,1\}^{\ell_{\mathcal{H}}}$.

4.4 Key Generation

Let us describe how the issuer chooses its public and secret keys. The public key will be accompanied by 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 anonymity property of DAA will hold.

1. The issuer chooses a RSA modulus $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$ such that p, p', q, and q' are all primes and n has ℓ_n bits.

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

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

$$Z := S^{x_z} \bmod n \ , \quad R_0 := S^{x_0} \bmod n \ , \quad \text{and} \quad R_1 := S^{x_1} \bmod n \ .$$

4. It produces a non-interactive proof *proof* that R_0, R_1, and Z were computed correctly, i.e., that $Z, R_0, R_1 \in \langle S \rangle$. That is, the issuer produces the "signature of knowledge"

$$\begin{aligned} \mathit{proof} = SPK\{(\alpha_0, \alpha_1, \alpha_z) : \\ Z \equiv S^{\alpha_z} \pmod n \ \wedge \ R_0 \equiv S^{\alpha_0} \pmod n \ \wedge \\ R_1 \equiv S^{\alpha_1} \pmod n\}(Z, R_0, R_1, S, n) \end{aligned}$$

   using binary challenges.

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 $\gamma := \gamma'^{(\Gamma-1)/\rho} \bmod \Gamma$ and choose a random element $\zeta_I \in_R \langle \gamma \rangle$.

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

If R_0, R_1, S, and Z are not formed correctly, it could potentially mean that the security property for the TPM/host do not hold. Also, if γ does not generate a subgroup of $\mathbb{Z}_\Gamma^*$, the issuer could potentially link different signatures. However, it is sufficient if the platform/host (i.e., owner of the TPM) verifies the proof that these quantities were computed correctly only

once; it could even be sufficient if one representative of platform users checks this proof.

We finally note that it is not important for the anonymity requirement that n be a product of two safe primes - the issuer is the only party whose security depends on this choice.

4.5 Obtaining Attestation (DAA-Join)

As input to the protocol, the TPM and the host receive the public key of the issuer and all the system parameters. We assume that the correctness of this public key is verified. Furthermore, the TPM has as input the endorsement secret key that allows it to authenticate itself to the issuer. The input of the issuer consist of its secret keys and the endorsement public key of the TPM. We assume that the issuer has verified prior to the protocol's execution that the endorsement public key of the TPM is properly certified and that the TPM-host pair are eligible to obtain attestation.

1. The TPM chooses a random $f \in \mathbb{Z}_\rho^*$, computes its pseudonym N_I with the issuer as $N_I := \zeta_I^f \bmod \Gamma$, and sends N_I to the issuer.

2. The issuer checks for all (f_0, f_1) on the rogue list whether $N_I \stackrel{?}{\neq} (\zeta^{f_0+f_1 2^{\ell_f}}) \pmod{\Gamma}$. 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.

3. Next, the TPM, the host, and the issuer run a protocol to obtain a signature on secret messages f_0 and f_1, where f_0 corresponds the ℓ_f low-order bits of f and f_1 the remaining ones (i.e., $f = f_0 + 2^{\ell_l} f_1$ and $0 \leq f_0 < 2^{\ell_f}$ holds). In this protocol, the issuer also needs to be convinced that

   - the secret messages are compatible with the pseudonym N_I and that
   - these messages are held within a valid TPM.

   Therefore, and as the role of the signature receiver is split between the TPM and the host, the protocol that the three parties run to issue the signature differs from the one presented in Section 3.8 and is as follows.

(a) The TPM chooses $v' \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, computes the quantity $U := R_0^{f_0} R_1^{f_1} S^{v'} \bmod n$, and sends U to the issuer.

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

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

with the issuer as the verifier.

(c) The TPM authenticates U to the issuer using its endorsement secret key (see remark below for details).

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

$$v'' := \hat{v} + 2^{\ell_v - 1} \qquad \text{and} \qquad A := \Big(\frac{Z}{U S^{v''}}\Big)^{1/e} \bmod n \ ,$$

and sends A and v'' to the host.

(e) To convince the host that A was correctly computed, the issuer as prover runs the protocol

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

with the host.

(f) The host verifies whether e is a prime that lies in $[2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$.

4. The host stores (A, e) and sends v'' to the TPM.

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

After this protocol the host and the TPM have together obtained a DAA-certificate , although each of them knows only parts of it.

In the protocol we required that the TPM prove to the issuer that the value U stems from the TPM that owns a given endorsement public key EK. Which, assuming that such a TPM would only authenticate value U that it had generated, 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 is in the same spirit as the protocol to "authenticate" the AIK in the towards the "Privacy CA" TCG TPM 1.1b specification.

As the DAA-Join protocol is based on the protocol given in §3.8, it can also only be run sequentially to guarantee security for the issuer. Of course, one could apply any of the techniques mentioned in §3.8 to obtain a protocol that can be run concurrently with many platform. However, a this had required substantial more code inside the TPM, it was preferred not to apply any of them but to restrict the number of platforms with which the protocol can be run concurrently. That is, the issuer can run the protocol concurrently with a small (logarithmic in the security parameter, e.g., 20-30) number of platforms in batches ensuring that the protocol with all TPM in a batch is either finished or aborted before it starts the next batch.

4.6 Proving Attestation (DAA-Sign)

Assume that the TPM and the host have obtained attestation, i.e., possess a DAA-certificate and want to convince a verifier of this fact. Before the protocol is run, the verifier specifies whether the protocol should be run w.r.t. a random base or a named base (cf. Step 1 of the protocol). The host specifies by a bit b whether the to be generated DAA-signature should be on an attestation identity key (AIK) the TPM has generated as message or on a message m chosen by the host. After this set-up, the protocol below is run. This protocol differs from the one described by Brickell et al. [4] in that it uses some optimizations as proposed by Camenisch and Groth [8].

The input to TPM are all system parameters, the issuer's public key, the TPM's output of the DAA-Join protocol, the bit b and, depending on b's value, a message m or an attestation identity key. The input to host are all system parameters, the issuer's public key, the host's output of the DAA-Join protocol, and, depending on the verifier's selection, the verifier's current base name $\mathtt{bsn}_V$. The input to the verifier are all system parameters, the issuer's public key, the bit b, a message m or, depending on b's value, an attestation identity key, and, depending on the verifier's selection, the verifier's current base name $\mathtt{bsn}_V$.

1. (a) Depending on the verifier's request (i.e., whether $\mathtt{bsn}_V \neq \perp$ or

not), the host computes ζ as follows

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

and sends ζ to the TPM.

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

2. (a) The 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 together they know a DAA-certificate and that N_V was computed from the secret messages contained in that certificate. Again, this could in principle be done just by the TPM using a version of the protocol discussed in the previous section to proof knowledge of a signature on a secret message and then that the secret message was used to compute N_V. However, as to save resources of the TPM, this task is distributed between the TPM and the host. More precisely, the TPM runs as prover the protocol

$$\begin{aligned} &PK\{(\nu, \varphi_0, \varphi_1) : \\ &\quad U \equiv S^\nu R_0^{\varphi_0} R_1^{\varphi_1} \pmod{n} \ \wedge\ N_V \equiv \zeta^{\varphi_0+\varphi_1 2^{\ell_f}} \pmod{\Gamma} \ \wedge \\ &\qquad\qquad \varphi_0, \varphi_1 \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+2}\} \end{aligned}$$

with the host as verifier. The host will then use the messages of this protocol and use then to generate the signature of knowledge

$$\begin{aligned} &SPK\{(\varepsilon, \tilde\nu, \varphi_0, \varphi_1) : \\ &\qquad Z A'^{-2^{\ell_e-1}} \equiv \pm A'^{\varepsilon} S^{\tilde\nu} R_0^{\varphi_0} R_1^{\varphi_1} \pmod{n} \ \wedge \\ &\qquad\qquad N_V \equiv \zeta^{\varphi_0+\varphi_1 2^{\ell_f}} \pmod{\Gamma} \ \wedge \\ &\quad \varphi_0, \varphi_1 \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+2} \ \wedge\ \varepsilon \in \pm\{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}+2}\}(b\|m) \ . \end{aligned}$$

This is done as follows.

(a) i. The host picks random integers $w \in \{0,1\}^{\ell_n+\ell_\varnothing}$ and computes $A' := AS^{-w} \bmod n$.

ii. The TPM picks random integers

$$r_v \in_R \{0,1\}^{\ell_v+\ell_\varnothing+\ell_\mathcal{H}}, \qquad r_{f_0}, r_{f_1} \in_R \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}},$$

and computes

$$\begin{aligned}\tilde{U} &:= R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_v} \bmod n \ , \\ \tilde{r}_f &:= r_{f_0} + r_{f_1} 2^{\ell_f} \bmod \rho \ , \text{ and} \\ \tilde{N}_V &:= \zeta^{\tilde{r}_f} \bmod \Gamma \ .\end{aligned}$$

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

iii. The host picks random integers

$$r_e \in_R \{0,1\}^{\ell'_e + \ell_\varnothing + \ell_{\mathcal{H}}} \quad \text{and} \quad r_w \in_R \{0,1\}^{\ell_e + \ell_n + \ell_\varnothing + \ell_{\mathcal{H}}}$$

and computes

$$\tilde{A}' := \tilde{U} A'^{r_e} S^{r_w} \bmod n$$

(b) i. Host receives the nonce n_v from the verifier and computes

$$\begin{aligned}c_h := H((n \| g \| g' \| h \| R_0 \| R_1 \| S \| Z \| \gamma \| \Gamma \| \rho) \| \zeta \| \\ A' \| N_V \| \tilde{A} \| \tilde{N}_V) \| n_v) \ .\end{aligned}$$

and sends c_h to the TPM.

ii. The TPM chooses a random $n_t \in \{0,1\}^{\ell_\varnothing}$ and 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)

$$\begin{aligned}s_\nu &:= \ r_v + c \cdot v \ , \quad s_{\varphi_0} := \ r_{f_0} + c \cdot f_0 \ , \quad \text{and} \\ s_{\varphi_1} &:= \ r_{f_1} + c \cdot f_1\end{aligned}$$

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

ii. The host computes (over the integers)

$$s_\varepsilon := \ r_e + c \cdot (e - 2^{\ell_e - 1}) \quad \text{and} \quad s_{\tilde{\nu}} := \ s_\nu + r_w + c \cdot (e \cdot w) \ .$$

4. The host sends the signature

$$\sigma := (\zeta, A', N_V, c, n_t, (s_{\tilde{\nu}}, s_{\varphi_0}, s_{\varphi_1}, s_\varepsilon)) \tag{3}$$

to the verifier.

5. The verifier first checks whether the values contained in σ are a valid signature of knowledge

$$SPK\{(\varepsilon, \tilde{\nu}, \varphi_0, \varphi_1) :$$
$$ZA'^{-2^{\ell_e-1}} \equiv \pm A'^{\varepsilon} S^{\tilde{\nu}} R_0^{\varphi_0} R_1^{\varphi_1} \pmod{n} \wedge$$
$$N_V \equiv \zeta^{\varphi_0+\varphi_1 2^{\ell_f}} \pmod{\Gamma} \wedge$$
$$\varphi_0, \varphi_1 \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+2} \ \wedge \ \varepsilon \in \pm\{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}+2}\}(b\|m) \ .$$

That is, the verifier

(a) computes

$$\hat{A}' := (ZA'^{-2^{\ell_e-1}})^c R_0^{s_{\varphi_0}} R_1^{s_{\varphi_1}} A'^{s_\varepsilon} S^{s_{\tilde{\nu}}} \bmod n \qquad \text{and}$$
$$\hat{N}_V := N_V^c \zeta_V^{s_{\varphi_0}+2^{\ell_f} s_{\varphi_1}}$$

and verifies whether

$$c \stackrel{?}{=} H(H(H((n\|g\|g'\|h\|R_0\|R_1\|S\|Z\|\gamma\|\Gamma\|\rho)\|\zeta\| A'\|N_V\|\hat{A}'\|\hat{N}_V)\|n_v)\|n_t)\|b\|m) \ .$$

and

$$s_{\varphi_0}, s_{\varphi_0} \stackrel{?}{\in} \pm\{0,1\}^{\ell_f+\ell_\varnothing+\ell_\mathcal{H}+1} \qquad \text{and} \qquad s_\varepsilon \stackrel{?}{\in} \pm\{0,1\}^{\ell'_e+\ell_\varnothing+\ell_\mathcal{H}+1}$$

all holds.

6. If ζ was derived from the verifier's base name, the verifier also checks whether $\zeta \stackrel{?}{=} (H_\Gamma(1\|\mathtt{bsn}_V))^{(\Gamma-1)/\rho} \pmod{\Gamma}$.

7. Finally, for all (f_0, f_1) on the rogue list, the verifier checks whether $N_V \stackrel{?}{\neq} (\zeta^{f_0+f_1 2^{\ell_f}}) \pmod{\Gamma}$.

4.7 Security Discussion

A full proof of security (and a formal model of DAA) is outside the scope of this chapter. Let us discuss here informally why the protocols described above meet our goals.

The informal properties that we wanted to achieve are unforgeability, anonymity, and rogue tagging (cf. 4.1). Unforgeability requires that a host

and TPM can only successfully run a DAA-Sign protocol if they had priorly obtained attestation, i.e., a signature by the attester/issuer on some secret messages, which in turned required to possess an endorsement certificate. This is satisfied because it can be shown that TPM and the host can only produce a DAA-Signature if possessing a DAA-certificate on some secret messages and these secret messages where used to compute N_V (in the prescribed way). Also note that the TPM's part of the DAA-certificate cannot be extracted from it (e.g., by a malicious host) by running the DAA-Sign or DAA-Join protocol, as the TPM only ever used these to compute pseudonyms N_I and N_V (from which the secret can only be extracted by computing discrete logarithms) and then as secrets in proofs of knowledge which provably reveal no information about the TPM's secrets f_0 and f_1. Now as these two secrets are an essential part of a DAA-certificate, the host cannot prove possession of a DAA-certificate without involving the TPM. It remains to argue that a DAA-certificate can only be obtained if the TPM possesses a certificate on its endorsement key pair. It is assumed that the TPM does not leak its endorsement secret key. Now, the mechanism used in the DAA-Join protocol ensures that the value U was indeed computed by the very same TPM that possesses the secret key corresponding to the certified endorsement public key. As the attester/issuer will only issue a certificate on messages committed in a U generated by an endorsed TPM, it is ensured that a certificate gets only issued to a TPM holding a valid endorsement certificate and, moreover, that the messages committed in U are only known to the TPM. Thus, we have argued unforgeability.

Let us consider anonymity. We have to show that a DAA-Signature cannot be linked to an execution of the DAA-Join protocol (i.e., to an issuer's transcript of the protocol) and that two DAA-Signatures cannot be linked provided they are produced w.r.t. a different named base ζ. Here, linking means that one cannot tell whether the two transcripts stem from the interaction with the same or different TPM-host pairs. First we note that, in the so-called random oracle model, all the information, i.e., the signature

$$\sigma = (\zeta, A', N_V, c, n_t, (s_{\tilde{\nu}}, s_{\varphi_0}, s_{\varphi_1}, s_\varepsilon)) \ ,$$

the host sends to the verifier is simulatable. That is, first of all by the zero-knowledge property of the proof of knowledge, the values $c, n_t, (s_{\tilde{\nu}}, s_{\varphi_0}, s_{\varphi_1}, s_\varepsilon)$ can be replaced by random values without that it can be noted. Next, the value A' can be replaced by a random element from $\langle S \rangle$ as A' is randomly distributed in $\langle S \rangle$ if computed correctly by the host without that it can be noted. Finally, ζ and N_V can be replaced by random values as well.

Now, this last change cannot be noted unless the Decisional Diffie-Hellman problem is easy.

Thus, given two DAA-Signatures made w.r.t. different bases ζ we could replace both of them by random values unnoticeable and therefore as being random signatures cannot be linked anymore. Similarly, an issuer's transcript of a DAA-Join protocol execution and a DAA-Signature cannot be linked as, again, the signature could be replaced by random values.

Let us finally consider rogue-tagging. This property requires that when secrets f_0 and f_1 have been extracted from a TPM (e.g., by physical extraction) together with the other parameters of a DAA-certificate, and that there secrets have been put on a black list, then it is recognizable if a TPM-host pair (or any other party) runs the DAA-Sign protocol with this certificate. That this property holds is easy to see: Given f_0 and f_1, anyone can compute $N_V := \zeta^{f_0+f_1 2^{\ell_f}} \bmod \Gamma$ and then check whether it matches the N_V received in a DAA-Signatures. Moreover producing such a signature requires knowledge of a certificate containing the messages corresponding to N_V, thus one could use the DAA-certificate but trying to use other messages to generate N_V.

4.8 Extensions

As we have seen only a part of the direct anonymous attestation protocol is run on the TPM and thus is "cast in stone", i.e., implemented in hardware. This allows for the modification of the non-TPM parts of the protocol to extend the direct anonymous attestation scheme in a number of directions.

For instance it is not hard to add attributes to the certificate, i.e., to have the issuer sign further (secret or non-secret) messages apart from f_0 and f_1. Let us discuss this extension. For simplicity we assume that the user of the platform wants to add the secret message m_2 as an attribute and that the issuer want to add the known message m_3 as an attribute. The requires that the issuer's public key further comprises values R_2 and R_3 randomly chosen from $\langle S \rangle$. Then the join protocol would need to be modified as follows: Before step 3d) of the Join protocol, the host would

i. choose $v'_h \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, compute $U_h := R_2^{m_2} S^{v'_h} \bmod n$ and and send U_h to the issuer;

ii. and prove to the issuer that U_h was computed correctly, i.e., run

$$PK\{(\mu_2, \nu'_h) : \quad U_h \equiv \pm R_2^{\mu_2} S^{\nu'_h} \pmod{n} \ \wedge \ \nu'_h \in \{0,1\}^{\ell_n+\ell_\varnothing+\ell_\mathcal{H}+2}\}$$

with the issuer as the verifier.

In the Steps 3c) and 3d), the issuer would needs to consider U_h and m_3, i.e., they would become the following steps.

3c) The issuer chooses $\hat{v} \in_R \{0,1\}^{\ell_v - 1}$ and a prime $e \in_R [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$, computes

$$v'' := \hat{v} + 2^{\ell_v - 1} \qquad \text{and} \qquad A := \big(\frac{Z}{UU_h R_3^{m_3} S^{v''}}\big)^{1/e} \bmod n \ ,$$

and sends A and v'' to the host.

3d) To convince the host that A was correctly computed, the issuer as prover runs the protocol

$$PK\{(\delta): \quad A \equiv \pm\big(\frac{Z}{UU_h R_3^{m_3} S^{v''}}\big)^{\delta} \pmod n\}$$

with the host.

All other steps of the Join protocol remain unchanged except that the host also needs to store (v'_h, m_2, m_3) along with (A, e).

Proving possession of attestation containing further attributes is similar to the original protocol, only that now the host has to extend the proof messages obtained from the TPM into a signature that takes into account the additional attributes. We leave it as an exercise to the reader to derive the protocol for doing this from the DAA-Sign protocol and the protocol described in §3.9 to prove possession of a CL signature. As in the latter protocol, the host could also provide to the verifier a commitment to an attribute contained in the attestation.

5 Applications of DAA

Besides the original purpose to authenticate a TPM's measurements, the DAA protocol has also other applications. Let us briefly discuss a two of them.

5.1 Binding Access-Credentials to a TPM

Assume an IT department of some company wants to protect access to the company's resources such as its VPN or accounting databases. Traditionally, these resources are protected by user name and password or, better, using public key cryptography with some protocol such as SSH. Unfortunately,

these credentials are vulnerable to malware such as viruses or phishing attacks. Also, a user might just reveal them to unauthorized persons. For instance, a user might share a subscription to, e.g., the Wallstreet Journal, with her friends by sharing her login-information. It seems that these problems can only be overcome by using some piece of tamper resistant hardware. Indeed, many companies protect access to VPN by replacing the password by a one-time password generated by a hard-ware token. However, this solution requires additional hardware to be distributed and furthermore does not scale very well.

The DAA protocol offers a solution to this that is also easy to deploy [6]: When our IT department gets shipped new laptops, they need only to enter the (public parts) of the endorsement keys of the TPM embedded into these laptops in their database together with the access rights the users of these laptops shall have. Then, when a user get the laptop, she only needs to initiate the DAA-Join protocol as a result of which she will get DAA-credential(s) to access all the applications she is granted access. The credentials for the different applications could either be distinguished by different issuer public keys or, better, by inserting one or more attributes into the credential specifying to which application(s) the credential will provide access. Later, if the user wants to access a service, she would initiate the DAA-Sign protocol, possibly revealing the attribute specifying the application she wants to access. As the user cannot only do this while having access to the TPM the credential was issued to, this effectively prevents malware and phishing attacks as well as social attacks where the users themselves leak credentials.

As the TPM can handle certificates from several issuers, such a systems also scales well, i.e., the same TPM could be used to protect the user's access to e-banking as well as her employer's VPN. One might argue that binding a user's credentials to a particular TPM has the drawback that the user can use her credentials only with a single platform. However, using attributes this can easily be overcome. The idea is as follows. The user would be required to register all her platforms with some third party whose sole purpose is to ensure that the same user cannot register too many platforms. In particular, the party does not need to be trusted by the user from a privacy point of view as we will see. This third party will then issue a DAA-credential to each of these platform such that all these credentials contain as attribute a number that is unique. The party need not to know this number, it just needs to ensure that it's unique to the user. So it could for instance be a random number that the user and the party jointly generate such that only the user learns it. Later, the party and the user would run the DAA-Join

protocol in such a way that this number will be embedded as an attribute.

When the user later wants to obtain a credential giving her access to some service, she sends the service provider a commitment to her unique number and proves to the service provider that she has obtained a DAA-certificate from the third party that contains the committed unique number. The service provider will then issue the user a CL-signature on that unique number as credential to access the service (*service-certificate*). Note that the service provider need not to learn this unique number (cf. §3.8) either. Now the user can use this service certificate with any TPM she registered. That is, to access the service she would provide the service provider a (fresh) commitment to her unique number and then prove that she possesses a DAA-certificate that contains as attribute the committed number as well as a CL-signature (service-certificate) by the service provider on that number. Thus, the service provider will be ensured that the user will not be able to share her credentials with her friends while the user can access the service anonymously with all her platforms.

5.2 Better Privacy Yet Detecting Rogues

Consider the privacy provided by the original DAA-Sign protocol as describe in §4.6. As we had discussed, the reason that the TPM sends the verifier the pseudonym N_V is to enable the verifier to assess whether or not the TPM is rogue. There are two different tests. The first one checks whether the claimed TPM at hand had used keys that are known to be rogue ones and the second one uses a frequency analysis, i.e., checks whether the keys used where used many times before. While the first check works for a base ζ_V chosen at random or for one that is derived from the verifier's base name, the second one does not work for a randomly chosen base. Indeed, the second test requires that the base ζ_V be fixed for a sufficiency long period of time. This however has the drawback that the privacy of legitimate TPM-host pairs is reduced as all their transactions with that verifier can be linked during this period of time. In particular, this would allow for profiling as proving ownership of a DAA-certificate and the actual payload (e.g., consuming a service) need to happen in same transactions. Luckily, one could use the technique described in the previous section to separate the frequency check and the payload transaction while retaining security [5]. The idea is that the platform (i.e., the TPM-host pair) first proves knowledge of a DAA-certificate w.r.t. to a long-term ζ_V. Then, if the frequency checks succeeds, a one-time anonymous credential is issued that is bound to the TPM as described above, where the one-time property is achieved by including as

secret attributes into the credential a random serial number and the name of the service provider of the payload (and possibly an expiration date). Then, the platform can contact the service provider for the payload transaction, show that it possess a DAA-certificate using a random base ζ and then proving possession of the one-time credential where there it needs to reveal the random serial number, the verifier's name and the expiration date and to prove that these are contained in the credential as attributes.

Therefore, the service provider is assured that the TPM is not a rogue while the platform is guaranteed (almost) the same privacy as if a random base was used.

6 Conclusion

In this chapter we have discussed the probably first privacy enhancing cryptographic protocol that is standardized. At the time of this writing, the first TPM chips implementing the DAA protocol become available. We have hinted that DAA is extensible and has some other applications apart from authenticating in a privacy friendly way the measurements made a TPM. We hope that the protocol and its extensions will make today's computer networks a more privacy friendly place and that many authentication needs can be satisfied while retaining privacy.

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 *Lecture Notes in Computer Science*, 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 *Lecture Notes in Computer Science*, pages 236–250. Springer Verlag, 1998.

[3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communication Security*, pages 62–73. Association for Computing Machinery, 1993.

[4] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proc. 11th ACM Conference on Computer and Communications Security*, pages 225–234. acm press, 2004.

[5] J. Camenisch. Better privacy for trusted computing platforxms. In P. Ryan and P. Samarati, editors, *European Symposium on Research in Computer Security — ESORICS 2004*, Lecture Notes in Computer Science. Springer Verlag, 2004.

[6] J. Camenisch. Protecting (anonymous) credentials with the trusted computing group's trusted platform modules V1.2. Technical Report TRxx-xx, IBM Reserach, Jan. 2005.

[7] J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In T. Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 331–345. Springer Verlag, 2000.

[8] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *proceedings of SCN '04*, volume 3352 of LNCS, pages 120–133, 2004.

[9] 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 *Lecture Notes in Computer Science*, pages 93–118. Springer Verlag, 2001.

[10] 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 *Lecture Notes in Computer Science*, pages 61–76. Springer Verlag, 2002.

[11] 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 *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2003.

[12] 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 *Lecture Notes in Computer Science*, pages 126–144, 2003.

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

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

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

[16] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[17] 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 *Lecture Notes in Computer Science*, pages 186–194. Springer Verlag, 1987.

[18] M. Fischlin. Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. In V. Shoup, editor, *CRYPTO '05*, 2005.

[19] 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.

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

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

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

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

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

[25] 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.

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