A Secure and Practical Approach for Providing Anonymity Protection for Trusted Platforms

Kurt Dietrich and Johannes Winter

Institute for Applied Information Processing and Communications
University of Technology Graz, Inffeldgasse 16a, 8010 Graz, Austria
{Kurt.Dietrich,Johannes.Winter}@iaik.tugraz.at

Abstract. Two different anonymisation schemes for Trusted Computing platforms have been proposed by the Trusted Computing Group - the PrivacyCA scheme and the Direct Anonymous Attestation scheme. These schemes rely on trusted third parties that issue either temporary one-time certificates or group credentials to trusted platforms which enable these platforms to create anonymous signatures on behalf of a group. Moreover, the schemes require trust in these third parties and the platforms have to be part of their groups. However, there are certain use-cases where group affiliation is either not preferred or cannot be established. Hence, these existing schemes cannot be used in all situations where anonymity is needed and a new scheme without a trusted third party would be required. In order to overcome these problems, we present an anonymity preserving approach that allows trusted platforms to protect their anonymity without involvement of a trusted third party. We show how this new scheme can be used with existing Trusted Platform Modules version 1.2 and provide a detailed discussion of our proof-of-concept prototype implementation.

1 Introduction and Background

One of the fundamental goals of Trusted Computing is to establish "trust-relationships" between computing platforms. These relationships are based on information about the integrity of the communication endpoints. This information can be provided by Trusted Platform Modules (TPMs) which form the core of every Trusted Platform and are shipped with millions of platforms (e.g. notebooks, PCs and servers).

TPMs are capable of storing and reporting platform integrity information and performing cryptographic operations. In order to exchange the configuration settings and integrity information of a platform, the TCG introduced the concept of remote attestation.

Remote attestation is based on integrity measurement and integrity reporting. The first one is done during the platform's boot where a hash of the loaded binary images is stored inside the TPM in so-called platform configuration registers (PCRs). This configuration can be reported to a remote platform by signing it - inside the TPM - with an attestation identity key (AIK). The verifying

M. Soriano, S. Qing, and J. López (Eds.): ICICS 2010, LNCS 6476, pp. 311–324, 2010.

platform can now validate the signature and the reported PCR values, thereby making a decision about the trust status of the signer platform [6]. However, along with this trust requirement comes the demand for anonymity in order to prevent tracking of platform and user activities. In order to achieve platform anonymity, the Trusted Computing Group which is responsible for developing and maintaining the Trusted Computing standards, has introduced two concepts:

The PrivacyCA (PCA) scheme was a first approach towards establishing anonymity for Trusted Platforms. This simple scheme is based on creating a temporary AIK and sending it to the PCA for certification. The verifier who receives data that was signed with a certain AIK and the AIK credential can verify the signature on the data and the credential but he only realizes that the credential was issued by a certain PCA - he is not able to identify the signing platform. By trusting the PCA, the verifier can also trust that the AIK was created and used inside a genuine TPM.

The Direct Anonymous Attestation Scheme (DAA) is an anonymous signature scheme based on group signatures and zero-knowledge proofs which allow TPMs to locally sign and certify an AIK without the involvement of a PCA [3]. Each TPM has its own private DAA-key that is certified by an issuer, however, for obtaining a credential from the issuer, the platform must *Join* a group managed by this certain issuer where the required DAA credentials are created, respectively issued. A platform can now create signatures on behalf of the group. A verifier can verify these signatures with the issuer's public-key without getting knowledge of the true signing platform.

However, both approaches rely on a trusted third party which is either a PCA or an issuer or group manager. This idea is acceptable as long as the interacting platforms are part of such a group. The group and the corresponding services could be established by a company or an official government institution. However, if you think of your private PC at home, which of these services would provide protection for your home platform? A company's anonymization service will unlikely provide such a protection for the company's employees' private computers in order to protect their transactions in their spare-time. A private computing platform would have to rely on either paid anonymization services which would add extra cost to the platform's owner in order to receive anonymity protection or it would have to rely on free and open anonymization services where a platform and its user have to trust that the information sent to the service is dealt with correctly. However, the platform owner has no influence and no hold on the correct treatment of the information and the availability of the service. This raises the general question of how two platforms that are not part of one of such groups can establish a connection and stay anonymous at the same time. Hence, it would be reasonable to have an anonymization scheme that does not rely on such a trusted third party like a PCA or DAA issuer and that does not produce extra costs for clients.

Although it is possible for the TPM vendor to play the role of the DAA issuer, it is practically not possible to do so without providing extra services (i.e. a DAA infrastructure). Such an infrastructure would include the issuer component where

the clients can obtain their credentials. Theoretically, it might be possible to ship TPMs with pre-installed credentials, however, in practice this is not the case. Furthermore, a revocation facility and a trusted-third-party service that checks and signs the issuer parameters is required - remember, before loading the signed issuer parameters into the TPM, their integrity and authenticity has to be checked what is done partly by the TPM and partly by the host platform. The signing party which signs the DAA parameters could also be the vendor, however, at the cost of an extra service and additional processing steps on the platform required for validating the parameters.

In order to address this problem, we turn our attention to *ring-signatures* which have been introduced by Rivest et al in the year 2001 [11]. This kind of signatures allows a signer to create a signature with respect to a set of public-keys. This way, a verifier who can successfully verify the signature, can be convinced that a private-key corresponding to one of the public-keys in the set was used to create the signature. However, which private/public key-pair was used is not disclosed. Moreover, these signatures provide another interesting property: the ring-signatures are based on ad-hoc formed groups or lists of public-keys which can be chosen arbitrarily by the signer and they do not, in contrast to the PCA scheme or DAA scheme, rely on a third party. This last property is the most interesting one as we want to exploit this property for our purpose.

However, such signatures can become large, depending on the number of contributing public-keys. Efficient ring-signature schemes have been proposed in [7] and [4]. Unfortunately, these schemes can not be applied for our purpose as we depend on the involvement of a TPM which we require to compute commitments and proofs for our approach.

1.1 Related Work and Contribution

Several publications address the topic of using ring-signatures for Trusted Computing systems: Chen et al proposed to use ring-signatures for hiding platform configurations [5]. Their approach aims at configuration anonymity which means that the signer proves that his platform's configuration is one out-of-n valid configurations. A verifier can check if the signer's configuration is a valid one, but the true configuration is not revealed. However, their paper does not focus on platform anonymity. In order to achieve platform anonymity, in addition to configuration anonymity, their approach still requires an extra anonymization scheme like PCA or DAA.

Tsang et al [15] discuss the application of ring-signatures in Trusted Computing. They investigate how this type of signatures could be used to implement a DAA scheme based on linkable ring-signatures. However, they do not provide a detailed discussion of their idea. Moreover, they rely on group managers that set up the scheme and its parameters, thereby reversing the advantage of the ring-signatures which allows to neglect third-parties.

In contrast to these two publications, we propose a scheme for platform anonymization in which trusted platforms do not require one of the above mentioned third-parties. Furthermore, we give a detailed discussion of how

ring-signatures based on the Schnorr signature algorithm [12] can be created using the TPM DAA commands. Therefore, we show how the Schnorr ring-signing scheme can be modified in order to meet the requirements of the TPM's DAA functionality. Moreover, we define a protocol that allows a TPM to obtain a credential from a TPM vendor which is further used in our approach.

Outline. In Section 2, we introduce our approach and define the basic requirements from a high level point of view. In Section 3, we discuss our idea using Schnorr based ring-signatures and how the TPM's DAA commands can be exploited to support our approach. We continue in Section 4 and discuss our prototype implementation and provide experimental results for our approach. Finally, Section 5 concludes the paper by discussing unsolved issues and giving an outlook on future work.

2 Highlevel Description of Our Approach

In this section, we give a high level discussion of our approach and define the following assumptions and definitions:

1. All TPMs are shipped with a unique RSA key-pair, the endorsement-key EK.
2. Moreover, we assume that the vendors of the TPMs have issued an endorsement certificate to the TPM's endorsement keys in order to prove the genuineness of the TPM.
3. Both the signing platform H and the verifying platform V have to trust the TPM and the TPM vendor.
4. An *endorsement-key* or EK denotes the endorsement key-pair (public and private part).
5. A *public-endorsement-key* or *public-EK* denotes the public part of an endorsement-key-pair.
6. An *endorsement-key-certificate* or *EK certificate* denotes a certificate that contains the public part of an endorsement key-pair.
7. A *schnorr-key* or *SK* denotes a schnorr key-pair (public and private part).
8. A *public-schnorr-key* or *public-SK* denotes the public part of a Schnorr-key-pair.
9. A *schnorr-key-certificate* or *SK-certificate* denotes a certificate that contains the public part of a Schnorr key-pair.

We take advantage of the fact that each TPM is part of a certain group right from the time of its production, namely the group that is formed from all TPMs of a certain manufacturer.

Our approach is based on a *ring-signature* scheme where the ring is formed by a set of public-SKs and closed inside the TPM of the signer. Therefore, we have to show a verifier that the public-SK of the signing platform is an element of a group of public-SKs and that the ring was formed inside a genuine TPM. If he can successfully verify the signature, the verifier can trust that the signature was created inside a TPM. An introduction to ring-signatures can be found

in [11] and [1] which we use as a basis for our trusted-third-party (TTP) less anonymization scheme.

For creating a signature, the signer chooses a set $S = (SK_0, .., SK_{n-1})$ of n public-SKs, that contribute to the signature. He computes the signature according to the algorithm discussed in Figure 1.

The ring is finally formed by computing the closing element inside the TPM. In typical Trusted Computing scenarios where remote attestation is used to provide a proof of the platform's configuration state, the signer generates an attestation-identity-key (AIK) with the TPM. This AIK is an ephemeral-key and can only be used inside the TPM for identity operations. In our scenario, the AIK is signed with the ring-signature which results in the signature σ on the AIK. Nevertheless, it is possible to sign any arbitrary message m with this approach.

The verifier can now validate the signature and knows that the real signer's public-SK is an element of the set S. As a consequence, the verifier knows that the signer was a trusted platform and that the ring was formed inside a TPM. However, the verifier can not reveal the real identity of the signer. How this is achieved in our approach is discussed in Section 3.

2.1 Discussion

A Signer H and verifier V have to trust the TPM and its vendor. The verifier V validates the public certificates of $SK_0, ..., SK_{n-1}$. If all certificates were issued by TPM vendors, the verifier knows that the signer platform is equipped with a genuine TPM from a certain vendor. Otherwise, he rejects the signature.
In contrast to the EKs which are pre-installed in the TPMs and certified by the vendors, SK are created dynamically in the TPM. Consequently, they have to be certified before they can be used for signature creation. How this is achieved and how SKs prove the genuineness of a TPM is discussed in Section 3.2.

The endorsement certificate and the SK-certificate cannot be linked to the TPM it belongs to, as it only provides information about the vendor of the TPM. This is true as long as the EK or the EK-certificate is not transmitted from a certain platform e.g. when used in a PrivacyCA or DAA scheme.

A typical Infineon EK-credential contains the standard entries: the public-EK of the TPM, a serial number, the signature algorithm, the issuer (which is an Infineon intermediate CA), a validity period (typically 10 years), RSA-OAEP parameters and a basic constraint extension [9]. The subject field is left empty. For our experiments, we created SK-test-certificates with according entries.

The design of the TPM restricts the usage of the EK which can only be used for decryption and limits its usage to the two aforementioned scenarios. In these schemes, the EK-certificate could be used to track certain TPMs as the PCA might store certification requests and the corresponding TPMs. If the PCA is compromised, an adversary is able to identify which TPM created certain signatures. This is not possible in our scheme, as rings are formed ad-hoc and no requirement for sending the EK from the platform it belongs to, exist.

An SK-certificate might be revoked for some reason. In this case, the signer must realize this fact before creating a signature. Otherwise, the signer could create a signature, including invalid SKs. Assuming that the signer uses a valid SK to create his signature, the verifier would be able to distinguish between valid (the signers) SK and invalid SKs. Consequently, the signer's identity could be narrowed down or in case all other SKs are revoked, clearly revealed.

A time stamp could be used to define the time of signature creation. The validating platform could then check if the certificate was revoked before or after the time of signing. However, this idea requires the signer to use Universal Time Code (UTC) format in order to eliminate the time zone information which could also be used to narrow down the identity of the signer.

One advantage of this approach is that the SKs may be collected from different sources. However, in order to keep the effort for collecting the SKs and managing the repositories low, a centralized location for distributing the SK-certificates could be reasonable. Such a location might be the TPM vendor's website but it is not limited to this location.

Our scheme can be applied in various use-cases where it is important to form ad-hoc groups with no dedicated issuer. Aside non-commercial and private usage scenarios, such groups, for example, often occur in peer-to-peer systems. Moreover, the scheme can be used according to Rivest's idea for whistle blowing [11].

3 Schnorr Signature Based Approach

In this section, we discuss our Schnorr signature based approach which is based on a publication from Abe et al, who proposed to construct ring-signatures based on Schnorr signatures [1] in order to reduce the size of the overall signature. In contrast to the approach from Rivest [11], the idea of Abe et al does not require a symmetric encryption algorithm for the signature creation and uses a hash function instead. This idea can be used for our approach with a few modifications of the sign and verify protocol. A major advantage of this approach is that we can use existing TPM 1.2 functionality to compute this kind of signatures. In order to do so, we can exploit the DAA *Sign* and *Join* protocol implementation of the TPMs v1.2.

Signature Generation. Let n be the number of public-SKs contributing to the ring-signature and H a hash function $H_i : \{0,1\}^* \Rightarrow \mathbb{Z}_n$. j is the index of the signer's public-key SK_j consisting of y_i, the modulus N_i and g_i with $N_i = p_i q_i$ and p_i, q_i are prime numbers. A signer S_j with $j \in (0, ..., n-1)$ has the private-key $f_j \in \{0,1\}^{l_H}$ and the public-key $y_j = g_j^{f_j} \mod N_j$.

The signer can now create a ring-signature on the message m by computing:

1. Compute $r \in \mathbb{Z}_{N_j}$ and $c_{j+1} = H_{j+1}(SK_0, ...SK_{n-1}, m, g_j^r \mod N_j)$
2. For i=j+1..$n-1$ and 0..$j-1$.
3. Compute $s_i \in \mathbb{Z}_{N_i}$ and $c_{i+1} = H_{i+1}(SK_0...SK_{n-1}, m, g_i^{s_i} y_i^{c_i} \mod N_i)$, if $i+1 = n$ then set $c_0 = c_n$.
4. Finally, calculate $s_j = r - f_j c_j \mod N_j$ to close the ring.

The result is a ring of Schnorr signatures $\sigma = (SK_0...SK_{n-1}, c_0, s_0, ..., s_{n-1})$ on the message m where each challenge is taken from the previous step.

3.1 Using a TPM 1.2 to Compute Schnorr Signatures

In our approach, we want to involve the TPM of the signer in order to close the ring by exploiting the TPM's DAA commands. A detailed explanation of the DAA commands and their stages can be found in the following Paragraphs of this Section.

Although the DAA scheme is based on Schnorr signatures, the TPM is not able to compute Schnorr Signatures a-priori. However, we can use the TPM_DAA_Sign and TPM_DAA_Join commands to compute Schnorr signatures for our purpose. Therefore, we extend the algorithm description with the stages that have to be gone through during the execution of the TPM commands:

A signature on the message m can be computed as follows: Let (g, N) be public system parameters, $y = g^f \mod N$ the public-key and f the private-key (Note that for computational efficiency, f is split into f_0 and f_1 inside a TPM). For simplicity reasons, we use a common modulus N and a fixed base g for all contributing platform's in our further discussions. M is 20 byte long nonce required for computing a DAA signature inside a TPM.

1. Let $L = y_i$ with $(i = 0..n-1)$ be a list of n public-keys including the signer's key that contribute to the signature and let j be the index of the signer's public-key y_j.
2. Execute TPM_DAA_Sign to stage 5 and retrieve $T = g^{r_0} \mod N$ from the TPM (see Table 1 for the DAA Sign command steps).
3. Compute a random M_{T_i} and $c_{j+1} = H_{j+1}(H(H(g||n||y_0||..||y_{n-1}||T))||M_{T_i})||1or0||morAIK)$
4. For $i = j+1..n-1$ and $0..j-1$.
   - Compute a random M_{T_i}, s_i.
   - Compute $c_{i+1} = H_{i+1}(H(H(g||n||y_0||..||y_{n-1}||e_i))||M_{T_i})||1or0||morAIK)$ with $e_i = g^{s_i} y_i^{c_i} \mod N$.
5. To close the ring, continue to execute the TPM_DAA_Sign command protocol:
   - Continue to stage 9 and send $c_{in} = H(g||n||y_0||..||y_{n-1}||e)$ with $e = T * y^{c_i}$ to the TPM which computes $c = H(c_{in}||M_{T_j})$ and outputs M_{T_j}.
   - Continue to stage 10 and send either:
   -- $b = 1$, m is the modulus of a previously loaded AIK
   -- $b = 0, m = H(message)$ to compute $c = H(c||b||m/AIK)$ (where $c_j = c$)
   - Continue at stage 11 and compute $s_j = r_0 + c_j f_0$ via the TPM
6. Abort the DAA protocol with the TPM and output the signature $\sigma = (c_0, s_0, .., s_{n-1}, M_{T_0}, ..., M_{T_{n-1}})$

Fig. 1. Schnorr Ring-Signature creation

Computing the Schnorr Ring-Signature. In order to compute a Schnorr signature, we can exploit the TPM_DAA_Sign command. Therefore, we start the protocol and execute stages 0 to 11 as defined in [14], however, for our purpose, only stages 2 to 5 and 9 to 11 are of interest.

Table 1. `TPM_DAA_Sign` Command Sequence

Stage	Input0	Input1	Operation	Output
0	DAA_issuerSettings		init	DAA_session handle
1	enc(DAA_param)	-	init	-
2	$R_0 = g$	n	$P_1 = R_0^{r_0} \mod N$	-
3	$R_1 = 1$	n	$P_2 = P_1 * R_1^{r_1} \mod N$	-
4	$S_0 = 1$	n	$P_3 = P_2 * S_0^{r_{\nu 1}} \mod N$	-
5	$S_1 = 1$	n	$T = P_3 * S_1^{r_{\nu 2}} \mod N$	T
.	.	.	.	.
.	.	.	.	.
9	c_{in}	-	$c' = H(c_1 \| \| M_T)$	M_{T_j}
10	b	m or AIK handle	$c_j = H(c' \| \| b \| \| m)$	c_j
11	-	-	$s_0 = r_0 + c_j f_0$	s_0

Table 1 shows the steps for running the DAA Sign protocol with a TPM. The TPM_DA
A_Sign command is executed in 16 stages by sub-sequent execution of the command.

It is not required to finish the *Sign* protocol and we can terminate the DAA session at stage 11 and leave out stages 12 to 15. Stages 6 to 8 have to be executed but the results can be ignored.

In order to use this approach, we had to modify the Schnorr signature generation and verification scheme: The TPM_DAA_Sign command requires a *nonce* from the verifier to get a proof for the freshness of the signature and computes $H(nonce||M_{T_j})$ where M_{T_j} is a random number generated inside the TPM. We do not require this proof and set $c_{in} = H(g||N||y_0||..||y_{n-1}||e)$ (with $e = g^{r_0} y_j^{c_{j-1}} \mod N$) in our scheme. However, the resulting value M_{T_j} has to be recorded as we require it to verify the signature. As a result, the TPM computes $c_j = H(H(c_{in})||M_{T_j})||1 or 0||m or AIK)$.

The rest of the stages may be ignored and the session can be closed by issuing a TPM_Flush_Specific command to the TPM. The resulting signature is $\sigma = (c_0, s_0, ..s_{n-1}, M_{T_0}, ..M_{T_{n-1}})$ plus the list of public-SKs $\{SK_0...SK_{n-1}\}$. The parameter $b = 0$ instructs the TPM either to sign the message m that is sent to the TPM or if $b = 1$ to sign the modulus of an AIK which was previously loaded into the TPM. In this case, m contains the handle to this key which is returned when the key is loaded by a TPM_LoadKey2 command [14]. The latter case is the typical approach for creating AIKs that may be used for remote attestation.

Verifying the Schnorr Ring-Signature. The signature $\sigma = (c_0, s_0, .., s_{n-1}, M_{T_0}, ..., M_{T_{n-1}})$ can now be verified as follows in Figure 2: The verification of the signature does not involve a TPM.

1. For i=0..n-1
2. Compute $e_i = g^{s_i} y_i^{c_i} \mod N$ and $c_{i+1} = H(H(H(g||N||y_0||..||y_{n-1}||e_i))||M_{T_i})||1or0||morAIK)$.
3. Accept if $c_0 = H_0(H(H(g||N||y_0||..||y_{n-1}||e_{n-1})||M_{T_0})||1or0||morAIK)$

Fig. 2. Schnorr Ring-Signature verification

Parameter Setup. Before executing the Join protocol, we have to generate the DAA parameters i.e. issuer public-key, issuer long-term public-key [14] which we require during the execution of the protocol to load our signature settings into the TPM. In order to compute the platform's public and private Schnorr key, we first have to commit to a value f_0 by computing $y = g_0^f \mod N$. This can be done executing the TPM_DAA_Join command: with the parameters: $R_0 = g, R_1 = 1, S_0 = 1, S_1 = 1$, a composite modulus $N = p * q$ where g is a group generator $g \in \mathbb{Z}_n$ and p, q prime values.

Table 2. TPM_DAA_Join Command Sequence

Stage	Input0	Input1	Operation	Output
0	DAA_count=0 (repeat stage 1)	-	init session	DAA_session handle
1	n	sig(issuer settings)	set- verify sig(issuer settings)	-
.	.	.	.	.
4	$R_0 = g$	n	$P_1 = R_0^{f_0} \mod N$	-
5	$R_1 = 1$	n	$P_2 = P_1 * R_1^{f_1} \mod N$	-
6	$S_0 = 1$	n	$P_3 = P_2 * S_0^{s_{\nu 0}} \mod N$	-
7	$S_1 = 1$	n	$y = P_3 * S_1^{s_{\nu 1}} \mod N$	y
.	.	.	.	.
24	-	-	E=enc(DAA_param)	E

After finishing the protocol, we have obtained the public Schnorr key y and the secret-key f_0 which is stored inside the TPM.

The DAA commands (as shown in Table 2) are executed in 25 stages by sub-sequently executing the command with different input parameters (*Input0*, *Input1*). Each stage may return a result (*Output*). Parameters that are marked with "-" are either empty input parameters or the operation does not return a result. Column *Stage* shows the stage, *Input0*, *Input1* the input data, column

Operation the operation that is executed inside the TPM and *Output* shows the result of the operation.

In stage 7 we can obtain the public-key $y = g^{f_0} \mod N$. Although they do not contribute to the public-key generation, the rest of the stages have to be run through in order to finish the *Join* protocol and to activate the keys inside the TPM.

The DAA_issuerSettings structure contains hashes of the system parameters (i.e. R_0, R_1, S_0, S_1, N) so that the TPM is able to prove whether the parameters that are used for the signing protocol are the same as the ones used during the *Join* protocol. A discussion how the issuer settings are generated is given in Section 4.

Security Parameter Sizes. We suggest the following sizes for the required parameters:

1. $l_h = 160$ bits, length of the output of the hashfunction H.
2. $l_n = 2048$ bits, a public modulus.
3. $l_f = 160$ bits, size of the secret key in the TPM.
4. $l_r \in \{0,1\}^{l_f + l_h}$ bits, random integers.
5. $l_g < 2048$ bits, public base $g \in \mathbb{Z}_n$ with order n.

3.2 Obtaining a Vendor Credential

One issue remains open: while all TPMs are shipped with an endorsement-key and an according vendor certificate, our Schnorr key does not have such a credential. Hence, we can

1. assume that TPM vendors will provide Schnorr credentials and integrated them into TPMs right in the factory.
2. obtain a credential by exploiting the DAA Join protocol.

While the first solution is unlikely to happen, the second one can be achieved with TPMs 1.2. For this approach, we have to use the public RSA-EK and the DAA_Join protocol from the TPM.

The credential issuing protocol runs as follows:

1. The TPM vendor receives a request from the trusted client to issue a new vendor credential
2. The vendor computes a nonce and encrypts it with the client's public-EK $EN = enc(nonce_I)_{EK}$
3. The client runs the Join protocol to stage 7 and sends EN to the TPM (see Table 2)
4. The TPM decrypts the nonce and computes $E = H(y||nonce_I)$ and returns E
5. The client sends (E, y, N, g) to the vendor who checks if (E, y) is correct.
6. The vendor issues a credential on the public Schnorr key y.

By validating the EK-certificate, the vendor sees that the requesting platform is indeed one of its own genuine TPMs. Moreover, the encrypted nonce can only be decrypted inside the TPM which computes a hash from y and $nonce_I$, therefore, the issuer has proof that U was computed inside the TPM which he issues a certificate.

3.3 Discussion

Experimental results show that the computation of a single sign operation involving a single public-key of the ring signature takes about 27 ms (on average) which is in total 27*(n-1) ms + sig_{TPM}. sig_{TPM} is the signing time of the TPM for the complete signature and n is the number of contributing keys. When computing a ring-signature with 100 public-keys, the overall time is about 3 seconds on average, making this approach feasible for desktop platforms. We used a Java implementation for our tests, hence, optimized C implementations (e.g. based on OpenSSL [13]) could increase this performance by a few factors. The verification of a ring signature takes about the same time as the signature creation. For details on the implementation see Section 4.

For the sake of completeness, we provide the performance values of our test TPMs, demonstrating the time required for a full DAA-Join command and the stages 0-11 of the DAA-sign command (see Table 3).

Table 3. `TPM_DAA_Sign` Command Measured Timings

Operation	Infineon	ST Micro	Intel
DAA Join:	49,7 s	41,9 s	7,6 s
DAA Sign Stages 0-11:	32,8 s	27,2 s	3,9 s

For the DAA *Sign* operation, we are only interested in the stages 0-11 (see Figure 1), hence we can abort the computation after stage 11. All measurement results are averaged values from 10 test runs. The Intel TPM is a more sophisticated micro controller than the ST Micro and Infineon TPMs and is integrated into the Intel motherboard chips which results in a tremendous performance advantage [10]. Details of the evaluation environment can be found in Section 4.

The slower performance of the Infineon TPM can be related to hard- and software side-channel countermeasures integrated in the microcontroller. These countermeasures are required to obtain a high-level Common Criteria certification such as the Infineon TPM has obtained[1].

The TPMs do not perform a detailed check of the *input0* and *input1* parameters, they only check the parameter's size which must be 256 bytes where the trailing bytes maybe be zero. Hence, it is possible to reduce the computation of

[1] `http://www.trustedcomputinggroup.org/media_room/news/95`

the commitment from $U = R_0^{f_0} R_1^{f_1} S_0^{\nu_0} S_1^{\nu_1} \mod n$ to $U = R_0^{f_0} \mod n$ where f_0 is the private signing-key by setting $R_0 = g$ and $R_1 = S_0 = S_1 = 1$.

A similar approach is used for the signing process. In stages 2 and 11 from Table 1 we compute the signature (c, s) on the message m. The message may be a hash of an arbitrary message or the hash of the modulus n_{AIK} of an AIK that was loaded into the TPM previously.

If a signer includes a certificate other than a EK_S certificate in his ring, the verifier recognized this when verifying the credentials. If the signer closes the ring with a decrypt operation outside the TPM, the signature cannot be validated as he obviously did not use a valid EK_S and the assumption that only valid $EK_S s$ may contribute to a signature is violated.

The originality of the TPM can be proven by the Schnorr EK-credential as the TPM vendor only issues certificates to keys that were created in genuine TPMs manufactured by himself. This is proven during the execution of the DAA Join protocol where the vendor sends a nonce to the TPM which he encrypted with the original endorsement-key.

One could argue that obtaining a new vendor credential for the public part Schnorr key is just another form of joining a group like in the DAA scheme. But remember that all TPMs are part of the group formed by the TPMs of a certain vendor right from the time of manufacturing. Consequently, it is not required and not even possible to join the group again. Hence, our modified join protocol is a way of obtaining a credential for the Schnorr key.

4 Implementation Notes

In order to obtain experimental results, we implemented a Java library exposing the required set of TPM commands to use the TPM's DAA feature (`TPM_DAA_Sign`, `TPM_DAA_Join`, `TPM_FlushSpecific`, `TPM_OIAP`). On top of these primitives we provide Schnorr ring-signatures. The implementation was done in Java 1.6 as the runtime environment supports the required cryptographic operations like RSA-OAEP encryption that is used for the EK operations and modular exponentiations [2] which are required for computing the Schnorr signatures. The OAEP encryption is required for EK operations which encrypt the DAA parameters that are created and unloaded from the TPM during the *Join* protocol. The parameters are loaded into the TPM and again decrypted during the *Sign* protocol. Note that before executing the *Join* protocol, the public-EK of the TPM has to be extracted from the TPM for example by using the TPM tools from [16].

Our test platforms (Intel DQ965GF, Intel DQ45CB and HP dc7900) were equipped with 2.6 GHz Intel Core 2 Duo CPUs running a 64bit Linux v2.6.31 kernel and a SUN 1.6 Java virtual machine. Communication with the TPM is established directly via the file-system interface exposed by the Linux kernel's TPM driver. Our tests were performed with v1.2 TPMs from ST Micro (rev. 3.11), Intel (rev. 5.2) and Infineon (rev. 1.2.3.16).

4.1 Signature Sizes

In a straight forward implementation the size of Schnorr ring signatures can grow relatively large. For self-contained Schnorr ring-signatures which do not require any online interactions on behalf of the verifier, the overall signature size can be given as $l_h + (l_{SK} + l_{M_T} + l_s) * n$ with n being the number of public keys in the ring.

We assume that a verifier demands to see the entire SK-certificates instead of just the SK- public-key. Assuming approximately 1.3 kilobyte per SK-certificate[2] and the security parameters given at the end of Section 3.1 this yields an overall signature size of $20 + (1300 + 20 + 256) * n = 20 + 1576 * n$ bytes.

The relatively large signature size can be reduced if the burden of fetching the SK-certificates is shifted to the verifier. We have investigated two simple strategies which can reduce the effective signature size to reasonable values, assuming that the verifier has online access to a SK-certificate repository.

An obvious size optimization is to embed unique SK-certificate labels, like certificate hashes, instead of the SK-certificates themselves into the ring signature. When using 20-byte certificate hashes as SK-certificate labels, the overall signature size can be reduced to $20 + (20 + 20 + 256) * n = 20 + 296 * n$ bytes. The downside of this optimization is that the verifier has to fetch all SK-certificates individually when verifying a signature.

Further reduction of the signature size is possible by embedding a label representing the ring itself instead of its underlying SK-certificates in the signature. When using this strategy, the signature size can be reduced to $20 + (20 + 256) * n + l_{label} = 20 + 276 * n + l_{label}$ bytes where l_{label} denotes the size of the label.

5 Conclusion

In this paper, we have proposed an anonymization scheme for trusted platforms that does not rely on specialized trusted third parties. Our approach is based on Schnorr ring-signatures which can be used with existing TPMs v1.2 without modifications of the TPM by well-thought exploitation of the TPM's DAA functionality.

We have shown that our scheme is feasible for desktop platforms and that even large signatures can be created and verified in acceptable time. The performance is only limited by the performance of available TPM technology which differs strongly between the various TPM vendors.

Future investigations could include approaches using the ECC based variants of the Schnorr algorithm. This will be of interest as soon as TPMs support elliptic curve cryptography. Moreover, an investigation of the approach whether it is feasible for mobile platforms or not could be done in the future.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This work has been supported in part by the European Commission through the FP7 programme under contract 257433 SEPIA.

[2] This is the size of a typical ASN.1 [8] encoded EK-certificate from Infineon which we used as a template.

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (2002)
2. Vanstone, S.A., Menezes, A.J., Van Oorschot, P.C.: Handbook of applied cryptography. CRC Press series on discrete mathematics and its applications. CRC Press, Boca Raton (1997); Includes bibliographical references (p. 703–754) and index
3. Brickell, E., Camenisch, J., Chen, L.: *Direct Anonymous Attestation*. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, pp. 132–145. ACM, New York (2004)
4. Chandran, N., Groth, J., Sahai, A.: Ring signatures of sub-linear size without random oracles. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 423–434. Springer, Heidelberg (2007)
5. Chen, L., Löhr, H., Manulis, M., Sadeghi, A.-R.: Property-based attestation without a trusted third party. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 31–46. Springer, Heidelberg (2008)
6. Dietrich, K., Pirker, M., Vejda, T., Toegl, R., Winkler, T., Lipp, P.: A practical approach for establishing trust relationships between remote platforms using trusted computing. In: Barthe, G., Fournet, C. (eds.) TGC 2007 and FODO 2008. LNCS, vol. 4912, pp. 156–168. Springer, Heidelberg (2008)
7. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 609–626. Springer, Heidelberg (2004)
8. Dubuisson, O., Fouquart, P.: ASN.1: communication between heterogeneous systems. Morgan Kaufmann Publishers Inc., San Francisco (2001)
9. Housley, R. (RSA Laboratories), Polk, W. (NIST), Ford, W. (VeriSign), Solo, D. Citigroup: Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile - rfc 3280 (2002)
10. Intel. Intel Desktop Board DQ45CB Technical Product Specification (September 2008), `http://downloadmirror.intel.com/16958/eng/DQ45CB_TechProdSpec.pdf`
11. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001)
12. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
13. The OpenSSL Project. OpenSSL. Programa de computador (December 1998)
14. Trusted Computing Group - TPM Working Group. TPM Main Part 3 Commands (October 26, 2006), Specification available online at `http://www.trustedcomputinggroup.org/files/static_page_files/ACD28F6C-1D09-3519-AD210DC2597F1E4C/mainP3Commandsrev103.pdf`; Specification version 1.2 Level 2 Revision 103
15. Tsang, P.P., Wei, V.K.: Short linkable ring signatures for e-voting, e-cash and attestation. In: Deng, R.H., Bao, F., Pang, H., Zhou, J. (eds.) ISPEC 2005. LNCS, vol. 3439, pp. 48–60. Springer, Heidelberg (2005)
16. TrouSerS The opensource TCG Software Stack (November 2, 2007)