Implementation Aspects of Anonymous Credential Systems for Mobile Trusted Platforms

Kurt Dietrich, Johannes Winter, Granit Luzhnica, Siegfried Podesser

Implementation Aspects of Anonymous Credential Systems for Mobile Trusted Platforms

Kurt Dietrich, Johannes Winter, Granit Luzhnica, and Siegfried Podesser
{Kurt.Dietrich, Johannes.Winter,
Siegfried.Podesser}@iaik.tugraz.at
Granit.Luzhnica@student.tugraz.at

Institute for Applied Information Processing and Communications
Graz, University of Technology
Inffeldgasse 16a, 8010 Graz, Austria

Abstract. Anonymity and privacy protection are very important issues for Trusted Computing enabled platforms. Protection mechanisms are required in order to hide activities of the trusted platforms when performing cryptography based transactions over the Internet, which would otherwise compromise the platform's privacy and with it the users's anonymity. In order to address this problem, the Trusted Computing Group (TCG) has introduced two concepts addressing the question how the anonymity of Trusted Platform Modules (TPMs) and their enclosing platforms can be protected. The most promising of these two concepts is the Direct Anonymous Attestation (DAA) scheme which eliminates the requirement of a remote authority but includes complex mathematical computations. Moreover, DAA requires a comprehensive infrastructure consisting of various components in order to allow anonymous signatures to be used in real-world scenarios. In this paper, we discuss the results of our analysis of an infrastructure for anonymous credential systems which is focused on the Direct Anonymous Attestation (DAA) scheme as specified by the TCG. For the analysis, we especially focus on mobile trusted platforms and their requirements. We discuss our experiences and experimental results when designing and implementing the infrastructure and give suggestions for improvements and propose concepts and models for - from our point of view - missing components.

1 Introduction

The anonymity of trusted platforms is a crucial topic in Trusted Computing. The use of common digital signature schemes requires complex public-key infrastructures and allows adversaries to track and identify certain signing platforms. In order to address this problem, the TCG has introduced two schemes to protect the anonymity of Trusted Platform Modules (TPMs) and with it the anonymity of their host platforms and users. One of these schemes - the Privacy CA (PCA) scheme - relies on standard public-key infrastructures and performs

the anonymization step in collaboration with a trusted third party. The other, from our point of view, more promising scheme is Direct Anonymous Attestation (DAA) which eliminates the requirement of a remote authority, but requires complex mathematical computations. Although large effort has been put into researching the efficient generation of anonymous signatures, little attention has been paid to the supporting infrastructures which are essential components for deploying anonymous credential technology in the field. When designing and developing infrastructures for the DAA scheme, one faces several obstacles.

The first problem is that different protocols exist. Different DAA schemes have been proposed, for example in [3], [10] and [5]. These schemes are either variations of the original RSA-based DAA protocol or schemes based on different cryptographic primitives. Second, different clients have to be supported. DAA client platforms may range from server systems and desktop PCs, down to mobile phones or smart-cards. Each of these platforms has different processing capabilities and, therefore, different requirements on the used DAA scheme. The third obstacle is credential revocation. It is easy to see that revoking an anonymous credential is a complex task. Common approaches either involve revocation authorities or rogue tagging mechanisms. However, rogue-tagging typically implies drawbacks like high resource requirements that may not be available on mobile platforms. Suitable infrastructures that take all these issues into account are missing. In order to address these problems, we have designed and implemented a DAA infrastructure. Therefore, we introduce a Join protocol specification that supports the different variations of DAA schemes mentioned before. Moreover, we introduce an online revocation check mechanism that allows us to move the rogue tagging check from the client to a trusted third party.

Although research has been done focusing on specific problems of DAA on mobile platforms ([7], [16]) investigations of the feasibility of DAA with all its aspects in the sense of a complete infrastructure with revocation etc. are missing. However, such infrastructures are especially interesting when taking new applications of DAA in the mobile world, such as anonymous authentication via NFC [1] or transport layer security (TLS) [8] into account. Therefore, we provide an investigation of the practicability and requirements of DAA and its infrastructure for mobile platforms. For our analysis, we have chosen the RSA-based DAA scheme as specified by the Trusted Computing Group (TCG) [14] as this is currently the only existing public version of a DAA scheme which is standardized by a public committee.

1.1 Related Work

Few publications can be found that address the topic of infrastructures for anonymous credentials. The most important one is the paper from Camenish et. al. [4] which deals with the design and the implementation of the *idmix* credential system prototype. The publication addresses protocols and credentials used in the *idmix* system which is related to DAA.

The remainder of this article is organized as follows. We discuss the overall architecture of our implementation and give detailed results of the server and

client components where we put special emphasis on the rogue tagging mechanism. Moreover, we discuss our proposed protocol with support for different types of clients and different schemes and we address the topic of credential revocation by proposing a modification of the online-certificate-status-protocol (OCSP).

2 Overview

The Direct Anonymous Attestation (DAA) scheme is a mechanism to provide trusted information to a challenger about a platform's integrity without compromising the platform's privacy. This can be achieved by applying an anonymous group signature on the attestation information provided by the TPM.

The DAA model involves three parties: the issuer or group manager, the signer (or client) and a verifier. The issuer manages the group and controls which platform may enter the group and creates a group credential for the entering platforms. Moreover, the issuer defines and publishes a set of group parameters which allows verifiers to validate signatures that have been created by group members.

Briefly summarized, an anonymous attestation scenario works as follows: on startup, the client platform which consists of a host (i.e. PC platform) and a TPM, executes a TCG based authenticated-boot and creates measurement values of the loaded software image, which are then stored in the TPM. The TPM now contains the current configuration of the platform. When the platform wants to prove this configuration to a remote platform, it generates a signature on these hash values with a so-called attestation key. The attestation key is a special RSA key which is created and used inside the TPM. The TPM ensures that the private parts of attestation keys acan not leave the TPM and that attestation keys are only used for signing configuration values. The platform sends the signed configuration values to the remote verifier which is then able to validate the local platform's configuration with the public part of the attestation key. Moreover, the remote platform can trust that the signature was generated in a genuine TPM by validating a DAA signature on the public attestation key.

For each attestation process, a new key has to be created and certified. This is where the DAA signature comes into play. Instead of sending the public-key to a Privacy CA every time, the TPM can locally certify its attestation key using an anonymous signature on the key. When using anonymous signatures to certify attestation key, the TPM first creates a temporary RSA key-pair. The public part of this temporary RSA key is locally signed with a DAA signature. Hence, the TPM is able to locally certify its temporary attestation key. This temporary attestation key is then used to create a signatures on the configuration values. When validating a signature over the configuration values, the remote verifier first has to verify the DAA group signature on the temporary attestation key. Before a TPM can create DAA signatures, it has to generate a private DAA-key f and execute the *Join* protocol with the group manager in order to receive a

group-credential for f. Once the TPM has joined a group, it is able to generate DAA signatures to locally certify public-key held by the TPM.

In order to analyze the efficiency of the scheme on mobile platforms, we developed an infrastructure that provides issuer, client and verifier components. Our DAA infrastructure is based on a client-server architecture consisting of different components which include an *issuer*, a *trusted-third party* (TTP) for long term certification of the issuer's public-key, a DAA status responder and a certificate authority (CA) that issues credentials for the TTP, the responder and the revocation authority. Moreover, the CA issues credentials for all players in the infrastructure to enable end-point authentication for the transport layer security (TLS) endpoints. Furthermore, it provides the corresponding revocation information for all issued certificates. All communication is established using the TLS protocol in order to provide authenticity of the endpoints and confidentiality and integrity of the transmitted data. Although our design and proposed protocol aims at supporting different clients, our current implementation focuses on desktop systems and mobile phones.

2.1 The Issuer

The issuer is responsible for generating the group credentials and for issuing the credentials to client platforms. Before a client can join the group, it has to be authenticated, allowing only authorized clients from *joining* the group. Moreover, the issuer has to perform a *rogue* and *revocation* check of the client's TPM. Therefore, the issuer maintains a list with Endorsement Keys (EKs) for all the clients, which are allowed to join. However, there are some cases when the issuer would allow arbitrary clients to join. Therefore, we introduce two modes of operation for the issuer:

1. **Public Access mode** - Clients can join even if their EK is not on the issuer's list. Their key will be put on the list on the fly during execution of the join protocol in order to be able to monitor who is in the group.
2. **Private Access mode** - Only clients that have an authorized EK on the issuer's list can join and request the corresponding group credentials from the issuer.

In our approach, we focus on the DAA protocol discussed in [10] which is available on many different platforms [2], [13], [7].

Issuer Setup Protocol Before the issuer can start to operate, it has to generate its group credentials which is done during the setup phase where all parameters required for the issuer are generated. The parameters include values for exclusive use by the issuer (e.g. issuer's private key) and public values like the issuer's public-key. The public values also include a non interactive proof showing that the issuer's parameters have been generated correctly. Parameters are generated according the following protocol:

1. The issuer chooses a modulus n with length l_n and primes p, q, p', q' such that: $n = pq, p = 2p' + 1, q = 2q' + 1$
2. Next, it chooses random integers $x_0, x_1, x_z \in [1, p'q']$ and $x \in [1, n]$ which will be used to generate the proof and computes $S = x^2 \bmod n$, $Z = S^{x_z} \bmod n$, $R_0 = S^{x_0} \bmod n$ and $R_1 = S^{x_1} \bmod n$
3. The issuer produces a non-interactive proof that S, Z, R_0 and R_1 are computed correctly. (see [3] for more details)
4. It generates rouge tagging parameters by choosing random primes ρ, Γ and $\gamma' \in_R Z^*_\Gamma$, satisfying $\Gamma = r\rho + 1$ such that r is an integer, $\rho \nmid r$, $2^{l_\Gamma - 1} < \Gamma < 2^{l_\Gamma}$, $2^{l_\rho - 1} < \rho < 2^{l_\rho}$ and $\gamma'^{(\Gamma-1)/\rho} \not\equiv 1 \bmod (\Gamma)$.
   Finally, the issuer calculates $\gamma = \gamma'^{(\Gamma-1)/\rho} \bmod \Gamma$
5. The public-key of the issuer is the tuple $(n, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$ and the private-key is the tuple (p', q').

In addition to the public-key, the issuer computes a proof that the parameters of the public-key are generated correctly. This proof can be used to verify that the $Z, R_0, R_1 \in \langle S \rangle$ and $S \in QR_n$ parameters are properly constructed. As the verification of the public-key proof is time and resource consuming, it may be delegated by a client to a trusted-third-party which verifies the proof and signs the group-key, thereby attesting the authenticity and the correctness of the key.

2.2 The Client

In our scenario, the mobile phone is a Java 2 MicroEdition (J2ME) application which runs on a mobile phone. These mobile platforms can be equipped with dedicated micro-controllers or software based TPMs as discussed in ([9], [6], [6]). After computing the secret key f, the client may execute the *join* protocol with the issuer in order to obtain its credentials. Once the client has received the credentials, it is able to create DAA signatures on behalf of the group. Any verifier can verify these signatures with the issuer's - respectively the group's - public-key. Prior to joining the group, the client has to verify the public-key of the issuer. It is very important to perform this step, since improperly generated issuer-parameters can cripple the anonymity protection of the DAA scheme [3].

Our client application delegates the computationally expensive parts of the issuer proof verification to a trusted third party (TTP). The steps done by the trusted third party are roughly: The TTP verifies the issuer's proof that $Z, R_0, R_1 \in \langle S \rangle$ and that S is a quadratic residue $\bmod\ (n)$. Then the TTP verifies that rouge tagging is set up correctly by checking whether ρ and Γ are primes, $\rho \mid (\Gamma - 1), \rho \nmid (\Gamma - 1)/\rho$ and $\gamma^\rho \equiv 1 \pmod{\Gamma}$. Finally, it checks if all parameters of the issuer's public-key have the required lengths.

Following these steps, the client has proof that all the issuer-parameters are computed correctly which implies that the security properties for the client still hold [3, page 9].

3 The Join Protocol

During the *Join protocol*, client and issuer exchange parameters. Some of these parameters are only temporary parameters which are used to derive other pa-

rameters and others are credentials which are finally issued by the issuer to be stored as client-key and group credential. Moreover, we have modified the protocol in order to support different (i.e. future versions and devices that cannot rely on a trusted host platform like smart-cards) versions of the DAA scheme. The *Join protocol* works as follows:

1. The client sends the join command, version and hash of its public EK.
2. The issuer receives the join command and tests if the requested version and DAA scheme are supported. When operating in *Private Access mode* (see section 2.1) the issuer now tests if the client's EK hash is on the list of allowed EK hashes. Regardless of the access mode the issuer requests the client to transmit its public EK.
3. The issuer chooses a random n_e of length l_Φ and encrypts it with the client's public EK. The encrypted n_e and the issuer's basename bsn are sent back to the client.
4. The client needs to show proof of possession of the public EK
   – The client's TPM decrypts n_e, picks a random ν' of length $l_n + l_\Phi$ and computes: $U = R_0^{f_0} R_1^{f_1} S^{\nu'}$ as well as $a_U = H(U \| n_e)$
   – The client sends $\zeta_I = (H_\Gamma(1 \| bsn_I))^{(\Gamma-1)/\rho}$ to its TPM.
   – The TPM verifies that $\zeta_I^\rho \equiv 1 \bmod \Gamma$ holds and computes $N_I = \zeta_I^{(f_0 + f_1 2^{l_f})}$
   – Finally the client sends U, cnt, N_I and a_U to the issuer.
5. Upon reception of a_U, the issuer computes $a'_U = H(U \| n_e)$
   – the client has proved that it is the owner of the EK if and only if $a'_U = a_U$ holds.
6. Next, the issuer checks for the rogue tagging using N_I (see section 4).
7. The issuer generates a random nonce n_i of length l_h and forwards it to the client.
8. The client needs to prove knowledge of f_0, f_1 and ν' to the issuer.
   – The TPM generates random numbers r_{f_0}, r_{f_1} of length $l_f + l_\Phi + l_H$ and $r_{\nu'}$ of length $l_n + 2l_\Phi + l_H$. It then computes $\tilde{U} = R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_{\nu'}} \bmod n$
   – Client sends $c_h = H(n \| R_0 \| R_1 \| S \| U \| \tilde{U} \| n_i)$ to its TPM
   – The TPM generates a random nonce n_t of length $l\Phi$ and computes the final hash c and values s_{f_0}, s_{f_1}, ν' as: $c = H(c_h \| n_t)$, $s_{f_0} = r_{f_0} + sf_0$, $s_{f_1} = r_{f_1} + sf_1$ and $s_{\nu'} = r_{\nu'} + s \cdot \nu'$
   – The client forwards $c, n_t, s_{f_0}, s_{f_1}, \nu'$ to the issuer.
9. The issuer verifies the computations done by the client and its TPM.
   – s_{f_0}, S_{f_1} must be of length $l_f + l_\Phi + l_H + 1$. $s_{\nu'}$ must have the length $l_n + 2l_\Phi + l_H + 1$.
   – The issuer computes $\hat{U} = U^{-c} R_0^{s_{f_0}} R_1^{s_{f_1}} S^{s_{\nu'}} \bmod n$ and the proof *if and only if* $c = H(H(n \| R_0 \| R_1 \| S \| U \| \hat{U} \| n_i) \| n_t)$ holds.
10. After verification of the client proofs, the issuer constructs a Camenisch-Lysyanskaya (CL) credential.
    – The issuer first generates a random number $\hat{\nu}$ of length l_ν and calculates $\nu'' = \hat{\nu} + 2^{l_\nu - 1}$. Then the issuer selects a random prime $e \in [2^{l_e - 1}, 2^{l_e - 1} + 2^{l'_e - 1}]$ and computes: $\Phi(n) = (p-1)(q-1)$ and $d = e^{-1} \bmod \Phi(n)$.
    – The issuer now calculates $A = (\frac{Z}{US^{\nu''}})^d \bmod n$ and sends it to the client.

11. The client challenges the issuer to prove correct computation of A. In order to do so, the client sends a nonce n_h of length l_Φ.
12. The issuer generates a random $r_e \in [0, p'q']$ and computes $\tilde{A} = (\frac{Z}{US^{v''}})^{r_e} \bmod n$, $c' = H(n\|Z\|S\|U\|A\|\tilde{A}\|n_h)$ and $s_e = r_e - c'd$
    – The response (c', s_e, A, e, ν'') is forwarded to the client.
13. The client verifies the issuer's response and obtains its CL credential.
    – To verify the issuer's response the client computes $\hat{A} = A^{c'}(\frac{Z}{US^{v''}})^{s_e} \bmod n$ and $c'' = H(n\|Z\|S\|U\|A\|\hat{A}\|n_h)$. The response is valid if and only if $c'' = c'$ holds and if e is prime with $e \in [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$.
    – Using ν'' the TPM computes $\nu = \nu' + \nu''$ and then stores (f_0, f_1, ν'') as private-key.

3.1 Prototype Implementation of the Join Protocol

No defined application protocol for the DAA join step has been defined either by the authors of [3] or by the TCG. The TCG Trusted Software Stack specification includes provisions for basic support of the DAA join and sign commands of the TPM [14]. The basic DAA support specified in [14] only consists of an API interface without any kind of provisions for the application level network protocols required to execute the DAA join sequence between an issuer and a client platform.

Trusted Channel between Issuer-Service and Clients For our prototype implementation, we developed a simple application protocol which allows the client to communicate with a DAA issuer over a TLS secured network connection. The application protocol discussed in this section describes a working prototype implementation which is intended to encourage a broader discussion of how a practical DAA issuer network service could look like.

The application protocol used by our prototype is a simple request response protocol based upon a series of simple ASN.1 messages exchanged over a trusted channel. In our prototype system, this trusted channel is established using a TLS protected TCP/IP network connection. Using TLS server authentication enables us to authenticate the DAA issuer service against the host willing to join the DAA group. The host can resort to standard PKI methods for verifying the identity of the DAA issuer based on its server certificate. On mobile platforms, the online certificate status protocol (OCSP) can be employed to delegate the possibly resource and communication intensive certificate validation step to a trusted third party.

Since different platforms have different strengths and computation powers and our implementation of the issuer should serve all of them, we support different versions of commands. As you can see from the join process section, the first message to be sent from the host to the issuer is the command message. This message includes the command which in that case is "join" and also the version of the join protocol.

In order to support multiple platforms and protocol we implement different forms of ASN.1 messages and differ them based on the used version. The first version would be e.g. "1.0" and the second one "1.1". Hence, whenever a client which can not handle the first option wants to join, it simply specifies which version it can handle or rather which it understands and then the issuer interacts with it in an appropriate form. Additionally the versioning scheme allows us to implement protocol changes in future while maintaining compatibility with existing clients.

4 Rogue Detection and Revocation

In order to detect and prevent compromised TPMs from joining or from signing messages, a rogue and revocation detection mechanism must be used. As mentioned in [3], if the secret of a TPM is revealed it is tagged as *rogue* and the values f_0 and f_1 are put on a blacklist. In order to check whether a TPM is rogue or not, the following steps have to be performed:

The host and the issuer *separately* compute $\zeta_I = (H_\Gamma(1\|bsn_I))^{(\Gamma-1)/\rho}$. Then the client's TPM computes $N_I = \zeta_I^{(f_0+f_1 2^{l_f})}$ and sends N_I to the verifier. Now the verifier checks for all pairs (f_0, f_1) on its blacklist whether $N_I = \zeta_I^{(f_0+f_1 2^{l_f})}$ holds, if a pair satisfying the equation is found the TPM is considered to be rogue.

The same procedure is used during the sign and verification process of a signature, allowing the verifier to check whether the TPM is rogue or not. Alternatively, the ζ_V (ζ_I) can be chosen at random. In this case, ζ serves only for rogue detection. But if we derive it from the bsn, we would have the same ζ for the same bsn, and it would be possible to link different actions of the platform.

In case the issuer acts as verifier and uses the same bsn, the platform may be identified and its transactions may be linked, since $\zeta_V = \zeta_I$ and can be linked to the identity, which should not be the case. The authors of [12] address this problem and propose a simple solution by changing the calculation of ζ_V to:

$$\zeta_V = (H_\Gamma(0\|bsn_V))^{(\Gamma-1)/\rho}$$

In this case, the $\zeta_V \neq \zeta_I$ so it is not possible to link the identity of the platform to the actions. The biggest problem here is how to gather information in order to assemble a blacklist. In [3][Page 14] it is mentioned that when a certificate (A, e, ν) and the values f_0 and f_1 are found, they are tested whether $A^e R_0^{f_0} R_1^{f_1} S^\nu \equiv Z \pmod{n}$ holds. If so, they are put on the blacklist. However, there is no mechanism to get the parameters f_0 and f_1. If somebody is able to extract them out of a TPM and uses them to impersonate another party there is no way to distinguish between the original platform and the impersonating platform. It is unlikely that the impersonator will publish the extracted secrets. Moreover, there is no way that a platform can check if its secrets have been extracted. Even if the owner of the platform somehow gets the information that the platform has been compromised, the secrets can not be extracted from the

TPM in order to perform a self-revocation. There is no command for that action because the secret is supposed to never leave the TPM even if it is requested by the user [15] (for example, if the user wants to revoke the DAA credential if he notices that the TPM keys are compromised). The only way a user can delete its credentials is by performing a take-ownership where new secrets (f_0, f_1) will be generated. However, if he did not extract and publish the old secrets, someone might use these secrets for signing on behalf of the group and there is no way to detect it since the pair (f_0, f_1) is not in the blacklist. Hence, the extracted credentials cannot be revoked and remain valid.

4.1 Online Credential Revocation Check

In order to shift the computational effort from the mobile client to a more resourceful platform we introduce a modification of the Online Certificate Status Protocol (OCSP) [11]. OCSP allows to retrieve the status of a certificate from an online source, the OCSP responder. To achieve this, the OCSP request structure contains the CertID field consisting of hash algorithm identifier, hash of the issuer's name and the issuer's key, and the serial number of the requested certificate, we want to obtain the status of. A status may be good, revoked or unknown. A status may be good, revoked or unknown. We slightly modified the RFC2560 *CertID* structure to include the basename, the pseudonym ζ and a reference the issuer's public-key, which is used to obtain the group parameters. Figrue 1 illustrates the information flow in our modified variant of the OCSP protocol.

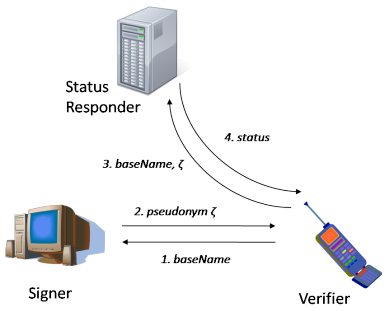


Fig. 1: Online Credential Status Reporting

Using the information provided in the modified *CertID* structure the responders checks if its black-list contains ζ and returns an appropriate OCSP status response. The authenticity and inbtegrity of the OCSP response can be verified using standard public-key signatures as discussed in RFC2560 [11].

5 Experimental Results

In this section, we discuss the results and setup of our infrastructure implementation. We tested the performance of the server (issuer) on a Sony Vaio

VGN-NR11Z/S notebook and the client on a Nokia 5800 Express Music cell phone, a Frescale i.MX51 development board and an MSI U135 netbook. Table 1 shows the main characteristics of the testing devices.

Device	Role	CPU	Clock Freq.	RAM	OS
Nokia 5800 Express Music	H	ARM11	434 MHz	128 MB	Symbian 60v5
Sony Vaio VGN-NR11Z/S	I	Intel Core 2 Duo	2.0 GHz	2 GB	Windows XP
i.MX51 EVK	H	ARM Cortex-A8	800 MHz	512 MB	Linux (Debian)
MSI U135 netbook	H	Intel Atom	1.2 GHz	1 GB	Linux (SuSE)

Table 1: Testing devices (Roles: H = Host, I = Issuer)

Overall performance evaluation of the DAA setup was done with the Sony Vaio notebook configured as server (issuer) and the Nokia mobile phone configured as client (host). The Freescale development board an the MSI netbook did not take part in the overall client/server performance evaluation. Instead the latter two devices were used to evaluate performance impact of using different Java virtual machine configurations as discussed later in section 5.2.

5.1 Overall client/server system performance

In the client/server performance test setup, both test devices were placed on the same local area network (LAN). Both the Nokia 5800 Express cell phone and the Sony Vaio notebook were connected to the test LAN using a wireless LAN access point. For this test. the issuer component was executed on a SUN Java 1.6 virtual machine configured with its installation default settings for Windows XP.

Modulus length	Join Time
1024 bit	10.85 s
1536 bit	19.56 s
2048 bit	32.52 s

(a) Join times with empty blacklist

Rouge TPMs	Time
100	33.57 s
1000	46.67 s
2000	59.76 s
5000	103.35 s
10000	187.20 s

(b) Join times with different black-list sizes

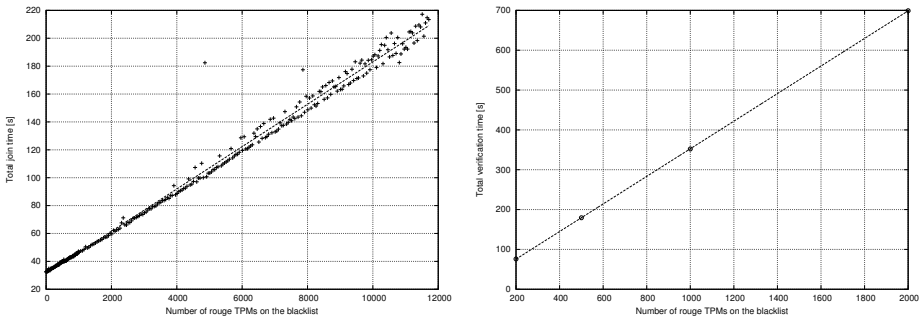
Rouge TPMs	Time
200	79.95 s
500	179.47 s
1000	352.11 s
2000	699.06 s
10000	187.20 s

(c) Rogue detection on a mobile phone

Fig. 2: Join and rouge detection times

Join Performance Results After the initial client and the server setup has been performed, the client can do the DAA join step. The performance results for joining a DAA group with empty blacklist shown below in table 2a include the network communication overhead. Since communication is done over network the results may vary in different connections with different strengths. These results were gained when performing joining with an empty blacklist. Hence, there is no rogue TPM on the list.

Rogue Detection Performance As discussed earlier the issuer has to compute a rouge tagging value $N_I = \zeta_I^{(f_0+f_1 2^{l_f})}$ for each pair (f_0, f_1) on its blacklist. A single N_I computation is very expensive, but overall cost increases linearly with the size of the blacklist. The more TPMs are on the list, the longer it takes to prove for any TPM whether it is rogue or not and the longer it takes to finish the join protocol. We measured the join process performance considering the effects of the rogue detection process. The tests were performed in a way that after each *Join* a new pair (f_0, f_1) is inserted into the blacklist for the next *Join*. In table 2b we show the time required for a join with rogue TPMs in the blacklist. The length of modulus for these measurements was 2048. As visible in figure 3a, there is a nearly linear dependency between the number of TPMs on the rouge list and the total time required for the joining process. Given those results the amount of time required for each additional rouge TPM corresponds to approximately 14.7 milliseconds or equivalently to 1.47 seconds per 100 rouge TPMs.



(a) Join performance over black-list size (b) Verification with rogue detection

Fig. 3: Join and verify performance

The biggest problem is that the rogue detection has to be performed when verifying signatures. Since a client should be able to verify signatures, it also should be able to check for rogue TPMs. We measured the execution time of the rogue detection depending on the number of TPMs in the blacklist on the Nokia mobile phone. The corresponding results can be found in table 2c and figure 3b.

Assuming medium to large blacklist sizes it clearly becomes evident that rouge detection is an expensive operation which might not be feasible on battery-powered devices like mobile phones. Moreover the level of user acceptance for signature verification processes in the order of minutes is at least questionable.

Proof Verification As previously discussed, a proof that the parameters of the issuer's public-key are generated correctly, should be generated by the issuer. In order to prove that $Z, R_0, R_1 \in \langle S \rangle$, a total of $3 \cdot 160 = 420$ relatively expensive calculations (verifications of bit-commitments) are required. Each of these calculations includes a modular exponentiation with an exponent $\in [1, p'q']$ and also a modular multplication. The proof verification step requires significant compu-

tational resources and takes significant time (≈ 99.44 s) even on the Sony Vaio platform used as issuer in our other experiments. On the Nokia mobile phone platform the verification time is far beyond any acceptability bounds (≈ 2399.24 s). The computing power of Java applications on mobile devices is clearly insufficient to verify this kind of proof. A viable solution for this case would be to delegate the verification step to a trusted third party.

5.2 Performance impact of different Java virtual machines

In the previous section we considered performance characteristics of a model realization of a DAA infrastructure based on a mobile phone acting as host and a notebook playing the part of the issuer. In this section, we briefly evaluate the impact of the Java virtual machine on the performance of the client side join process. In contrast to section 5.1 we ignore any server side computations and network overhead. Measurements shown in this section were performed using a special version of the client application which just instruments the client-side computations.

Platform	Java Virtual Machine	Average join time (client)	Deviation
Freescale i.MX51	OpenJDK/Zero	8.591 s	0.019 s
Freescale i.MX51	OpenJDK/Shark (mixed)	6.087 s	0.510 s
Freescale i.MX51	OpenJDK/Shark (interpreted)	142.742 s	0.274 s
Intel Atom N450	OpenJDK/Client (interpreted)	40.312 s	0.209 s
Intel Atom N450	OpenJDK/Client (mixed)	3.512 s	0.025 s

Table 2: Average timing values for client-side computations of the join process

We tested four combinations of Java virtual machines and just-in-time compiler settings which are intended to model typical JVMs found on mobile platforms. In order to produce comparable results for the tested platforms we used OpenJDK 1.6.0 as Java runtime environment. All cryptography related operations, including big number support was implemented with IAIK's JCE-ME[1] library. We tested the mixed-mode and interpreted Java VM configurations given in table 2. Figure 4 plots client-side computation times for repeated execution of the join process from within the same virtual machine instance. Table 2 gives average values for computation times. We decided to sample execution times for one particular VM configuration inside a loop from within a single VM invocation. This decision allows us to evaluate initial delays due to class loading and just-in-compilation. Three of the four VM configurations depicted in 4 exhibit relatively constant timing behavior with some minor jitter caused by other operating system processes interfering with our measurements. Only the OpenJDK "Shark" virtual machine exhibits great variations in join process execution times, caused by the relatively expensive just-in-time compilation steps done during the first few join process executions. As evident from figure 4 there is a huge performance gap almost in the range of one order of magnitude between

[1] See `http://jce.iaik.tugraz.at/sic/Products/Mobile-Security/JCE-ME`

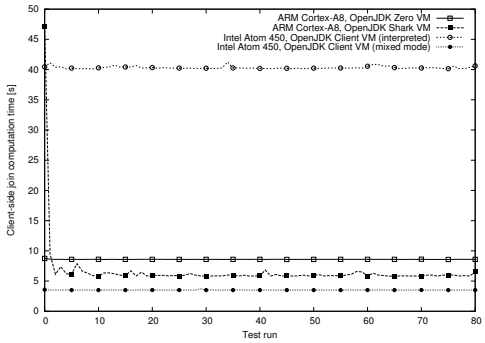


Fig. 4: Join computation performance for different Java VM configurations

the JIT (mixed mode) and the non-JIT (interpreted only) configurations on the Intel Atom platform. Interestingly the performance gap between the non-JIT and the JIT virtual machines on the ARM platform is by far smaller.

6 Conclusion

Based on the overall performance results of our implementation shown in section 5.1 we conclude that the most time-consuming part of the DAA protocol is the rogue detection. Even if rogue detection can be done relatively fast on desktop PCs and servers it still negatively affects the verification process which might be done on battery-powered mobile devices. Recalling from the test of the rogue TPMs, the time required by rouge checking process increases lineraly with the size of the black-list. For larger black-lists the time spent for rouge detection easily exceeds the actual signature verification time by orders of magnitude. The worst part is, that rouge detection has a big influence on mobile phones, even for small blacklists with sizes in the order of 30 rouge TPMs. Therefore, we conclude that delegating the validation of the rogue status to a third party is an unavoidable requirement.

Another open problem with rogue detection are the mechanisms and protocols used to report corrupted TPMs. The current implementation of DAA in TPM 1.2 implicitly anticipates compromise of the DAA private key as *only* revocation reason. Currently there is no method that enables a user to voluntarily report corruption of his platform without knowing the f value guarded by the TPM. Moreover the proof verification process required to check the validity of an issuer's public key requires impractical amount of storage and computational resources on mobile phones. This problem can be solved by off-loading the proof verification to a trusted third party. Finally the results from section 5.2 clearly show the impact of the choice of Java virtual machine on the performance of our purely Java-based prototype implementation.

Acknowledgments. 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.

References

1. Berna, S., Yalcin, O. (eds.): *Radio Frequency Identification: Security and Privacy Issues - 6th International Workshop, RFIDSec 2010, Istanbul, Turkey, June 8-9, 2010, Revised Selected Papers*, Lecture Notes in Computer Science, vol. 6370. Springer (2010)
2. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: *Anonymous credentials on a standard java card*. In: CCS '09: Proceedings of the 16th ACM conference on Computer and communications security. pp. 600–610. ACM, New York, NY, USA (2009)
3. Brickell, E., Camenisch, J., Chen, L.: *Direct Anonymous Attestation*. Conference on Computer and Communications Security Proceedings of the 11th ACM, Washington DC (5), 132–145 (November 2004)
4. Camenisch, J., Van Herreweghen, E.: *Design and implementation of the idemix anonymous credential system*. In: CCS '02: Proceedings of the 9th ACM conference on Computer and communications security. pp. 21–30. ACM, New York, NY, USA (2002)
5. Chen, L., Page, D., Smart, N.P.: *On the Design and Implementation of an Efficient DAA Scheme*. Cryptology ePrint Archive, Report 2009/598 (2009), `http://eprint.iacr.org/`
6. Dietrich, K.: *An Integrated Architecture for Trusted Computing for Java Enabled Embedded Devices*. In: STC '07. pp. 2–6. ACM, New York, NY, USA (2007)
7. Dietrich, K.: *Anonymous Credentials for Java Enabled Platforms: A Performance Evaluation*. In: INTRUST. pp. 88–103 (2009)
8. Dietrich, K.: *Anonymous Client Authentication for Transport Layer Security*. In: Communications and Multimedia Security. pp. 268–280 (2010)
9. Dietrich, K., Winter, J.: *Implementation Aspects of Mobile and Embedded Trusted Computing*. In: Chen, L., Mitchell, C.J., Martin, A. (eds.) TRUST. Lecture Notes in Computer Science, vol. 5471, pp. 29–44. Springer (2009), `http://dblp.uni-trier.de/db/conf/trust/trust2009.html#DietrichW09`
10. Mitchell, C.: *Direct Anonymous Attestation in Context*. In: Trusted Computing (Professional Applications of Computing). pp. p. 143–174. IEEE Press, Piscataway, NJ, USA (2005)
11. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP* (1999)
12. Smyth, B., Ryan, M., Chen, L.: *Direct Anonymous Attestation (DAA): Ensuring Privacy with Corrupt Administrators*. In: Stajano, F., Meadows, C., Capkun, S., Moore, T. (eds.) ESAS. Lecture Notes in Computer Science, vol. 4572, pp. 218–231. Springer (2007), `http://dblp.uni-trier.de/db/conf/esas/esas2007.html#SmythRC07`
13. Sterckx, M., Gierlichs, B., Preneel, B., Verbauwhede, I.: *Efficient Implementation of Anonymous Credentials on Java Card Smart Cards*. In: 1st IEEE International Workshop on Information Forensics and Security (WIFS 2009). pp. 106–110. IEEE, London,UK (2009)
14. Trusted Computing Group: *TCG Software Stack (TSS) Specification Version 1.2 Level 1* (6 January 2006), part 1: Commands and Structures
15. Trusted Computing Group: *TPM Main Specification Level 2 Version 1.2, Revision 103*. Tech. rep., Trusted Computing Group (26 October 2006)
16. Wachsmann, C., Chen, L., Dietrich, K., Löhr, H., Sadeghi, A.R., Winter, J.: *Lightweight Anonymous Authentication with TLS and DAA for Embedded Mobile Devices*. In: ISC. pp. 84–98 (2010)