Anonymous RFID Authentication Using Trusted Computing Technologies

Kurt Dietrich

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

Abstract. Anonymity protecting mechanisms are an important part of any Trusted Computing platform. They provide protection of a platform's anonymity and, consequently, protection of the privacy of the platform's owners. As Trusted Computing technologies have been introduced on mobile and embedded systems and more and more mobile devices are equipped with Near Field Communication (NFC) modules, the question arises whether the supported anonymization mechanisms can be used efficiently for anonymous authentication for NFC enabled applications. However, state-of-the-art technologies like the Direct Anonymous Attestation scheme require complex mathematical computations that put high requirements on the processing power of the signer's device which are typically not available on resource constrained devices like smart-cards. In this paper, we analyze how the Direct Anonymous Attestation protocol can be used for anonymous authentication in NFC scenarios and we propose an approach that allows a practical use of this technology in real-world scenarios.

Keywords: Mobile Trusted Computing, Secure Element, Direct Anonymous Attestation, NFC.

1 Introduction

Today, many desktop computers are equipped with Trusted Platform Modules (TPMs). They provide support for anonymity preserving technologies which have been a major topic in Trusted Computing since its beginnings. In order to achieve anonymity protection for trusted platforms, two different concepts have been introduced by the Trusted Computing Group (TCG): the *PrivacyCA* (PCA) scheme and the *Direct Anonymous Attestation* (DAA) scheme. Both allow a trusted platform to hide its identity when doing operations over the internet. Both schemes aim at preserving a platform's anonymity, however, with different methods. The first scheme is based on remote certification of public-keys on a regular basis, where the platform is required to create a temporary key-pair for each transaction prior to performing this transaction. The public part of the temporary key-pair has to be sent to the PCA which certifies it. A verifier who receives data that was signed with this temporary key is able to verify the

S.B. Ors Yalcin (Ed.): RFIDSec 2010, LNCS 6370, pp. 91–102, 2010.

signature and the authenticity of the key with the certificate from the PCA. As each transaction requires a different key-pair and a new certificate, a verifier is not able to link the signature to the originating signer platform. However, this approach has some severe drawbacks. Every temporarily created key-pair has to be sent to the PCA, which consequently, has to be permanently available. Moreover, the PCA can operate in different modes. In the first mode, it does not record any certification requests. In the second mode, it records and stores all certification requests from the single platforms. Therefore, the PCA is able to link the issued certificates to the requesting platforms. This fact opens a security leak in case the PCA gets compromised. If an adversary gets hold of the recorded information, it could use this information to link transactions and signatures to the single platforms.

In order to overcome these problems, the TCG introduced another scheme: the Direct Anonymous Attestation scheme. It omits the requirement for a remote party to certify the temporary keys as the keys can be certified locally, on a platform, by a TPM. This scheme is based on a group signature scheme and Zero-Knowledge proofs of knowledge, allowing each platform to create signatures on behalf of a group which can be verified by a single group-public-key. The advantage of this scheme is that no on-line connection to a third party is required. Moreover, different signatures created by the same platform cannot be linked to this certain platform - not even by the group manager which is responsible for issuing group credentials to each platform in the group.

In case the issuer becomes compromised, an adversary only gets the knowledge that a certain platform is part of the group that is managed by this specific issuer. The adversary is not able to link any signature that has been created before the compromission or any signature that will be created afterwards, to a single platform.

However, this scheme requires complex computations, making it hard to use on resource constrained devices like smart-cards. Furthermore, the scheme requires a secure storage on the device for protection of the group and DAA credentials as they require authorized access for usage and must not be copied or moved to a different platform.

In contrast to the PCA approach, the DAA scheme allows local certification of public-keys, thereby omitting the requirement of certifying keys by a remote PCA. This fact makes the DAA scheme especially interesting for mobile devices, as it does not require the mobile to open remote connections. Moreover, with the raise of NFC technology, DAA like anonymous signature schemes become an interesting technology for preserving anonymity for NFC enabled applications. They can be used as a basic building block for providing anonymity in e-ticketing- and mobile-payment- scenarios. Furthermore, they may be used for anonymous access control in order to prevent tracking and profiling of user activities.

1.1 Related Work

Different approaches have been published to integrate DAA functionality on smart-cards: The basic idea of splitting the computations between a resource

limited micro-controller and a powerful host has first been discussed by Brands in [3]. Bichsel et. al. [2] and Sterckx et. al. [15] analyzed implementations of variations of the DAA signature protocol on JavaCards. Both publications give performance results of their implementations which show that a practical use of the DAA scheme requires a powerful host to execute the host side computations of the protocol. This statement is supported by Balasch [9] who implemented a DAA scheme on an AVR micro controller. He concludes that the DAA scheme can only be practically used in combination with a resourceful host. As shown in [2], the computation of an entire signature takes about 16,55 seconds with a modulus length of 1984 bits on a JCOP v2.2/41 JavaCard. Balasch requires 133.5 seconds on an 8 bit micro-controller (with 1024 bit modulus) and Bichsel 450 seconds [2], however, Bichsel and Balasch only take the computations located inside the TPM into account.

Another publication concerning anonymous signatures on mobiles is [7]. Dietrich proposed to execute the DAA computations on a Java virtual machine which is executed in a protected environment on the mobile. This protection is achieved by the ARM TrustZone CPU extension [1] which supports the separated execution of trusted and un-trusted code in a *secure* and a *non-secure world*. This technique can be used as a building block for hosting TPM functionality [20] and for storing and using DAA credentials in a protected environment. The benefit of this approach is that the software running in the protected world can take advantage of the computing power of the main CPU so that anonymous signatures can be computed in sufficient and user acceptable time [7]. However, this special CPU extension is currently only available on some high-end smart phones using ARM 11 CPUs. For cheaper, low-cost phones, which typically employ ARM 9 or ARM 7 CPUs and which are the typical and, therefore, more widespread user devices, this technology is not available.

Furthermore, all of the publications mentioned before omit *rogue tagging*. Rogue tagging is a mechanism to detect malicious TPMs and is, therefore, an important feature when using anonymous credentials.

1.2 Our Contribution

In this paper, we discuss a design that allows the application of the DAA scheme for contactless, anonymous authentication in NFC enabled scenarios, with respect to secure storage, authorized access to the DAA credentials and sufficient performance. We show how the DAA protocol has to be extended in order to realize a system which is able to compute authentication information with a reasonable performance that can be applied in real-world scenarios. In addition, we use off-the-shelf devices to demonstrate our efforts in order to support this statement. Our approach can be used to enable privacy protecting technologies on low-cost devices which were previously only available on cost-intensive, high-end devices.

In our approach, we extend the idea of Dietrich [6] who proposes to place the TPM functionality into an on-board smart-card, also known as *Secure Element* (SE). A SE basically provides the same functionality as a common smart-card

or SIM-card, thereby establishing two major requirements of Trusted Platform Modules, namely *shielded locations* and *protected capabilities* ([8], [18]).

The rest of this article is organized as follows: we provide background information on mobile Trusted Computing and the DAA scheme. We discuss our proposed approach and modifications and give experimental performance results followed by a discussion of our prototype implementation. Finally, we summarize our results and propose ideas for further research.

2 Background

2.1 Mobile Trusted Computing

The Trusted Computing Group has published a specification that defines requirements for mobile TPMs [17]. This specification allows the mobile TPM vendors a great amount of flexibility with regard to concrete implementations. A mobile TPM may be implemented as a dedicated micro-controller or as a software module, depending on the security features provided by the target platform. Such features may include isolation of the TPM functionality to establish shielded locations and protected capabilities as required by the TCG. Examples of such isolation techniques are the L4-micro-kernel operating system, the ARM TrustZone processor extension or, as in our approach, an on-board smart-card, the secure element.

2.2 Direct Anonymous Attestation

The DAA protocol is basically a group signature scheme based on Zero-Knowledge proofs [4]. Instead of showing a credential to a verifier like in common public-key infrastructures (PKIs), the creator of a DAA signature computes a proof that it is in possession of certain group credentials. A detailed discussion of the DAA protocol is out of scope of this document and we refer to [7] and [12] for further information. Hence, we focus on a high-level discussion of the basic protocols *DAA Join* and *DAA Sign*.

The DAA Join Protocol: Before a device can create anonymous signatures on behalf of a group, it has to *join* the group and obtain the group parameters (S_0, S_1, n and R) from the group manager. Moreover, the TPM creates the secret keys f and ν' that are created during the join phase. The key ν' is separated into two parts, ν'_1 and ν'_2 as the cryptographic co-processor of the JavaCard does not allow exponents (ν' is 2128 bits) to be larger than the modulus (2048 bits). During the join process, the client proves knowledge of f to the group manager. The group manager computes the credentials (A, e, ν'') where e is a random prime and ν'' a random integer and computes a proof that A was generated correctly. The client verifies the proof on A and verifies that e is a prime in a certain interval. After successful execution of the Join protocol, the client has obtained the credentials $(A, e, \nu = \nu' + \nu'')$ which represent a signature on the key f that is protected by the TPM. Moreover, the TPM has obtained a value ν

and the platform has obtained a value ν'' which allows the TPM together with the host to create a DAA signature σ on a message m.

The DAA Sign Protocol: After a device has obtained the required credentials from the group manager it may create signatures on behalf of this group. In the *DAA Sign protocol*, host and TPM compute a signature σ with the credentials obtained during the *Join protocol*.

Algorithm 1. DAA Signature Creation

Input: $R, S_0, S_1, Z, \nu_0, \nu_1, n, f, \Gamma, \rho$.
Output: $\sigma = (T, c, n_t, s_{\bar{\nu}}, s_e, s_f, s_{\nu_0}, s_{\nu_1})$

1. The *host* selects a random w and computes: $T = A * S^w$, note: $S = S_1 * S_2^{2^{l_s}}$ and $\nu = \nu_1 + \nu_2 * 2^{l_s}$ for $l_s = 1024$
2. *Host* and *TPM* compute the "signature of knowledge":
3. The *TPM* computes: $\tilde{T}_t = R^{r_f} S_1^{r_{\nu_1}} S_2^{r_{\nu_2}} \bmod n$ with random r_f and r_ν
4. The *host* computes: $\tilde{T} = \tilde{T}_t T^{r_e} S^{r_{\bar{\nu}}} \bmod n$
5. and: $c_h = H(n \| R \| S_1 \| S_2 \| Z \| T \| \tilde{T} \| \Gamma \| \rho \| n_\nu)$
6. The *TPM* selects a random n_t and computes computes $c = H(H(c_h \| n_t) \| m)$ and $s_f = r_f + c * f$, $s_{\nu_0} = r_{\nu_0} + c * \nu_0$, $s_{\nu_1} = r_{\nu_1} + c * \nu_1$,
7. The *host* computes $s_e = r_e + c * (e - 2^{l_e - 1})$ and $s_{\bar{\nu}} = s_\nu + r_{\bar{\nu}} - cwe$
8. Finally, the *host* assembles the signature $\sigma = (T, c, n_t, s_{\bar{\nu}}, s_e, s_f, s_{\nu_0}, s_{\nu_1})$

Algorithm 1 shows the basic steps for computing a DAA signature. n is an RSA modulus with $n = p * q$ an $p = 2p' + 1, q = 2q' + 1$. S_1 is a random quadratic residue $\bmod n$ that generates the bases $R, S_2, Z \in S_1$. For further details on the parameters we refer to [12].

Rogue Tagging: In order to identify malicious TPMs, the TCG has introduced the so-called *rogue tagging* mechanism. Algorithm 2 shows the basic steps for computing the rogue tagging pseudonym of the signer.

Algorithm 2. DAA Rogue Tagging

Input: $basename, \Gamma, \rho$.
Output: ζ.

1. The verifier selects a random *basename* and sends it to the signer
2. The *TPM* computes $\zeta = H(basename)^{(\Gamma - 1)/\rho} \bmod \Gamma$

If a TPM is compromised and its private-key f becomes publicly known, a verifier can identify the compromised TPM via the pseudonym ζ [19]. The verifier holds a list with all known keys f, and computes the pseudonym values of these keys with respect to Algorithm 2 and the basename he provided. If ζ is on the computed list, the verifier knows that the signer was a malicious TPM.

3 Our Approach

All existing approaches ([2], [15], [9]) clearly show that current smart-cards do not provide sufficient performance for computing RSA based DAA signatures. Hence, it is inevitable to include a host that contributes the computation of such a signature. This can either be done by providing a host with adequate processing capabilities or by providing a host that controls and manages the pre-computation of such signatures. Idle phases of the device or the SE can be used to generate RSA key-pairs - which we address as ephemeral authentication keys (EAKs) from now on - which can then be certified by a DAA signature. Mobile phones, equipped with either SIM-cards or secure elements provide the ideal platform for such an approach.

We investigate two approaches how anonymous signatures can be used for authentication: in the first approach, the signature is computed entirely in the SE. In this case, the application on the host initiates the pre-computation of the keys and the signature creation. The algorithm listed in Algorithm 1 is executed entirely inside the TPM. The pre-computation of the certified EAK key-pairs can be executed on the card without further involvement of the host. In the second approach, the signature computation (see Algorithm 1) is partially computed inside the TPM and partially on the host. Details of the implementation and performance results can be found in Section 4.

However, in both approaches, the rogue tagging value cannot be computed in advance as it depends on a *basevalue* created by the verifier. Although a DAA signature, according to the specification [18], contains the actual signature and the computed rogue tagging value, both computations are rather independent cryptographic operations (only the computed hash c - see 1 contains the rogue parameters). Hence, they can be computed separately.

3.1 Anonymous Authentication Scenario

Figure 1 shows the application of our approach in a basic authentication scenario.

The mobile platform pre-computes a set of n EAK key-pairs (steps 1-3), certifies the public parts with DAA signatures[1] and stores the private parts of the keys either in the EEPROM of the SE or encrypts it and un-loads it to the host device. The public-keys and their credentials (i.e. the DAA signatures) are stored on the mobile platform. The same is true for the DAA credentials (f, ν_0 and ν_1, R_0, S_0, S_1). By loading different credentials, the TPM may create DAA signatures on behalf of different groups it has joined before.

A user can now use these keys and the NFC module on the phone to prove his authorization against an NFC terminal without revealing his identity. Before sending a request to the terminal, the mobile loads a certified EAK key into the TPM, either from the EEPROM or from the mobile device (steps 4-5). The terminal computes and sends a nonce r_n and *base* for rogue detection to the

[1] In Trusted Computing enabled application scenarios, the standard exponent 65537 is used. Hence, only the RSA modulus is signed and transmitted when required.

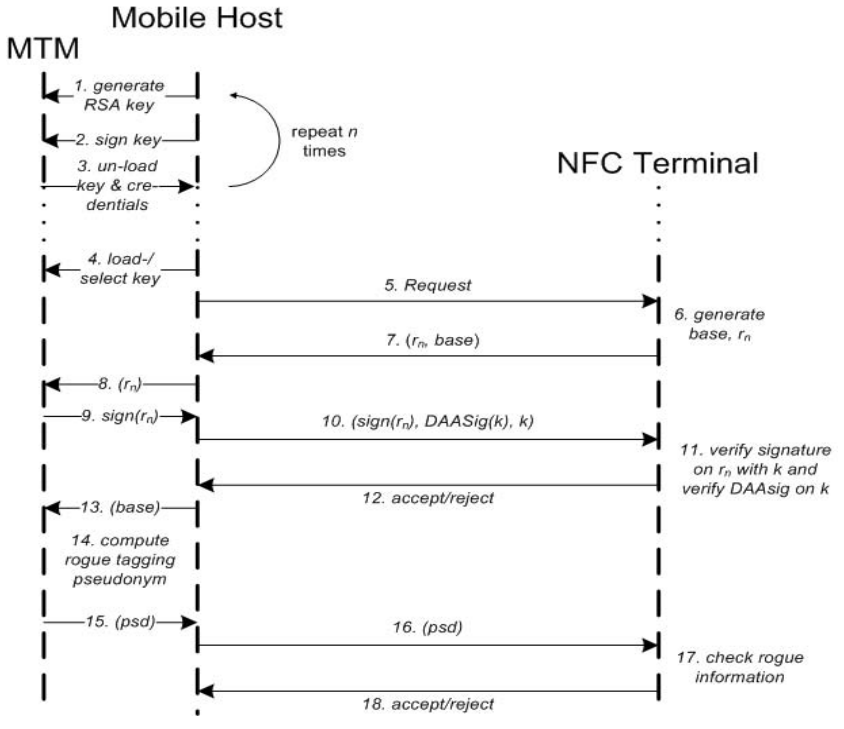

Fig. 1. Authentication Protocol Sequence

mobile (steps 6-7). The mobile forwards r_n to the TPM which signs r_n with the previously loaded EAK key. The mobile device forwards the signature $sign(r_n)$ on r_n, the public EAK key k and the DAA signature $DAASig(k)$ on this key to the terminal (step 10). The terminal verifies $DAASig(k)$ with the issuer's public-key (step 11) and continues the protocol if the verification succeeds. In steps 13-15, the TPM computes the pseudonym $psd = H(base)^{\Gamma-1/\rho} \bmod \Gamma$ which is verified by the terminal as discussed in [12].

If all verifications succeed, the terminal has the information that the requestor is a member of a certain group - namely the group represented by a certain issuer and its public-key - and that the used TPM is not on a list of compromised TPMs. However, the terminal has no information about the identity of the platform or its owner.

4 Implementation Aspects

For our experiments, we used a Nokia 6212 NFC mobile phone. This phone is equipped with a Giesecke & Devrient SmartCafe smart-card as SE. Our secure element based TPM uses this smart-card which provides a JCOP41 v2.2.1 runtime environment. The TPM commands and the DAA computations are handled

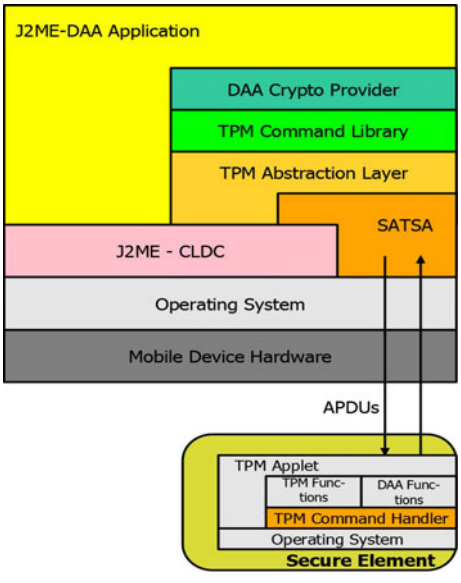

Fig. 2. Architecture Overview

by a JavaCard applet that is installed on the smart-card. Figure 2 shows our concept. The host application uses a TPM command library to issue commands which are sent to the SE via application protocol data units (APDUs).

The host part is implemented as a Java2MicroEdition (J2ME) [16] application that allows the installation of mobile applications on the phone. Moreover, we take advantage of the Security and Trust Service API (SATSA) [13] respectively of JSR 257 the *Contactless Communication API* [14] which allows our application to communicate with the card applet via APDUs. This approach, however, requires that the J22ME application is signed with a code signing certificate from Versign or Thawte.

For the DAA support in the TPM, we require several TPM commands and structures as defined in [18] as well as support for different algorithms. The JavaCard 2.2.1 environment does not provide support for implementing cryptographic protocols. We follow the ideas from [15] and [2] concerning algorithm implementations on JavaCard. For example, the modular exponentiation can be computed via the RSA cipher algorithm and modular multiplication via transformation into a binomial form, $((a*b) \bmod n = \frac{(a+b)^2-a^2-b^2}{2} \bmod n)$, the *hmac* algorithm has to be implemented in Java, reducing the overall performance when computing the integrity check of incoming TPM commands.

Our minimum implementation of the DAA scheme requires the following TPM commands and TPM structures on the host and TPM side:

1. a protocol for authorization: TPM_OIAP plus session handling,
2. the TPM_DAA_Join() command
3. the TPM_DAA_Sign() and TPM_DAA_Sign_Init() commands

4. TPM_FlushSpecific() and TPM_Terminate_handle commands for aborting the computation during one of the stages and freeing the resources inside the TPM.
5. TPM_DAA_Issuer_Struct. This structure holds the issuer parameters.
6. two containers for symmetric keys

For unloading the RSA key-pairs, the corresponding DAA signature and the DAA credentials, our TPM generates two symmetric keys k_0, k_1, one for encrypting the data and one for computing an integrity check on it. The TCG specification allows to use symmetric or asymmetric encryption for this purpose. In our approach, we use symmetric cryptography for encryption and - this is different to the TCG specification - a symmetric key for integrity protection to detect modifications of the encrypted authentication keys and DAA parameters when they are stored on the device.

4.1 The Pre-computation Step

The TCG specifies two commands *TPM_DAA_Join* and *TPM_DAA_Sign* which are executed repeatedly in different stages [18]. For simplicity reasons, we reduce these stages to a single stage.

Table 1 shows the measured performance values. The first row shows the values when the computation is split between host and TPM. The first column shows the required time for command handling which includes the computation and verification of *hmac* integrity checks on the command data and its transmission to the TPM. The second column shows the time consumed for computing the host part and the third column shows the time required for computing the TPM part inside the SE. The last column shows the overall result of all single operations.

Table 1 shows a slight performance advantage when computing the entire DAA signature in the TPM[2]. For the first approach, the JavaCard applet that includes the TPM command handler, the cryptographic algorithms and the DAA functionality, requires about 5284 bytes in the EEPROM of the card.

Table 1. Performance comparison of the DAA sign approaches

Command handling	Host	Secure Element	Total
1,1 s	23,8 s	4,8 s	29,7 s
1,4 s	-	26,0 s	27,4 s

A detailed performance analysis of the single cryptographic steps (i.e. random number generation, hash operations, modular exponentiation etc.) of the JavaCard applet and the Java application on the host can not be given in this

[2] For the interested reader, a DAA signature computation on an Infineon 1.2 TPM requires approximately 38 seconds.

version of the paper due to its length restrictions. Note that all performance measurements are average values that were estimated by 100 executions of the single operations.

4.2 The NFC Authentication Step

For the actual authentication over the NFC channel, we can use the certified EAK-keys from the pre-computation step. As shown in Figure 1, the terminal sends a nonce to the mobile/TPM which basically applies an RSA signature according to PKCS#1.5 [11] on the nonce which takes approximately 1 second.

Table 2. Performance of the authentication process

Command Handling	**Nonce singing**	**Rogue Tagging**	**Total**
1,0 s	1,3 s	1,1 s	3.4 s

Moreover, the TPM computes the rogue tagging parameter which is basically a single modular exponentiation which also takes approximately 1 second. Hence, the total time required for authentication requires 3,4 seconds.

Parameter lengths. In our prototype implementation, we use the following parameter lengths:

Table 3. Parameter lengths in number of bits

l_n	l_s	l_e	l_f	l_ν
2048 bits	1024 bits	368 bits	160 bits	2536 bits
l_ϕ	l_{r_w}	l_{r_ν}	l_{r_f}	l_Γ
80 bits	2128 bits	2228 bits	400 bits	2048 bits

5 Conclusion and Future Work

In this paper, we discuss how Trusted Computing based technologies can be used for anonymous authentication for NFC and RFID applications. Autonomous contactless smart cards are too constrained to achieve a satisfying performance, hence, we employ mobile phones equipped with NFC modules for our approach. We investigate two approaches, the first splitting the computation of such signatures between a resource constrained TPM and a more powerful host platform and the second, using solely the TPM to compute the entire DAA signature. Furthermore, we show how this approach can be used to circumvent the demanding computational requirements for computing the mathematical complex Direct Anonymous Attestation signatures in order to provide a feasible approach to show that our idea can be used in real-world scenarios. We achieve this by using the mobile phone for pre-computing and certifying authentication keys.

For generating experimental results, we use off-the-shelf mobile phones that are equipped with secure elements to host the TPM functionality and which are connected to NFC modules, allowing a practical implementation of our prototype.

We also propose modifications in the DAA protocol flow in order to speed-up the computations on the mobile platform.

Future investigations should include DAA schemes based on elliptic curve cryptography as discussed in ([10], [5]). ECC based schemes clearly show a performance advantage over the RSA based variant. Although support for ECC is provided by JavaCard vendors, adequate support for developing complex DAA protocols based on ECC is not yet available on current JavaCard platforms. Another interesting aspect for mobile devices is power consumption. How much power is drained from a device's battery depends strongly on the executed operations. Consequently, an analysis of the power consumption when computing DAA signatures is of great interest for future investigations.

Acknowledgements. This work has been supported by the European Commission through project FP7-2009.1.4(c)-SEPIA, Grant agreement number 257433.

References

1. ARM Ltd: TrustZone Technology Overview. Introduction, `http://www.arm.com/products/esd/trustzone_home.html`
2. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard java card. In: Proceedings of the 16th ACM conference on Computer and communications security, CCS 2009, Chicago, Illinois, USA, pp. 600–610. ACM, New York (2009)
3. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge (2000)
4. Brickell, E., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: Proceedings of the 11th ACM conference on Computer and communications security, CCS2004, Washington DC, USA, pp. 132–145. ACM, New York (2004)
5. Chen, L.: A daa scheme requiring less tpm resources. Cryptology ePrint Archive, Report 2010/008 (2010), `http://eprint.iacr.org/`
6. Dietrich, K.: An Integrated Architecture for Trusted Computing for Java Enabled Embedded Devices. In: Proceedings of the 2007 ACM workshop on Scalable trusted computing, STC 2007, pp. 2–6. ACM Press, New York (2007)
7. Dietrich, K.: Anonymous Credentials for Java Enabled Platforms. In: Chen, L., Yung, M. (eds.) INTRUST 2009. LNCS, vol. 6163, pp. 88–103. Springer, Heidelberg (2010)
8. Dietrich, K., Winter, J.: Implementation aspects of mobile and embedded trusted computing. In: Chen, L., Mitchell, C.J., Martin, A. (eds.) Trust 2009. LNCS, vol. 5471, pp. 29–44. Springer, Heidelberg (2009)
9. Balasch Masoliver, J.M.: Smart Card Implementation of Anonymous Credentials. Master's thesis, K.U.Leuven, Belgium (2008)
10. Page, D., Chen, L., Smart, N.P.: On the design and implementation of an efficient daa scheme. Cryptology ePrint Archive, Report 2009/598 (2009), `http://eprint.iacr.org/`

11. RSA Labs. PKCS1 v2.1: RSA Cryptography Standard (2001)
12. Mitchel, C.: Direct Anonymous Attestation in Context. In: Trusted Computing (Professional Applications of Computing), Piscataway, NJ, USA, pp. 143–174. IEEE Computer Society Press, Los Alamitos (2005)
13. SUN Community process: Java Specification Request (JSR-177): Security and Trust Services API. Specification (September 2004), http://jcp.org/en/jsr/detail?id=177
14. SUN Community process: Java Specification Request (JSR-257): Contactless Communication API. Specification (October 2004), http://jcp.org/en/jsr/detail?id=257
15. 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), London,UK, pp. 106–110. IEEE Computer Society Press, Los Alamitos (2009)
16. SUN Community process JSR 139. J2ME(TM) Connected Limited Device Configuration (CLDC) Specification 1.1 Final Release. Specification (March 4, 2004), http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html
17. Trusted Computing Group Mobile Phone Working Group. TCG Mobile Trusted Module Sepecification Version 1 rev. 1.0 Specification (June 12, 2007), https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf
18. Trusted Computing Group - TPM Working Group: TPM Main Part 3 Commands. Specification (October 26, 2006), http://www.trustedcomputinggroup.org/files/static_page_files/ACD28F6C-1D09-3519-AD210DC2597FE4C/mainP3Commandsrev103.pdf Specification version 1.2 Level 2 Revision 103
19. Trusted-Computing-Group-TSS-Working-Group. TCG Software Stack (TSS) Specification Version 1.2 Level 1. Specification (January 6, 2006), https://www.trustedcomputinggroup.org/specs/TSS/TSS_Version_1.2_Level_1_FINAL.pdf; Part1: Commands and Structures
20. Winter, J.: Trusted computing building blocks for embedded linux-based arm trustzone platforms. In: Proceedings of the 3rd ACM workshop on Scalable trusted computing, STC 2008, pp. 21–30. ACM, New York (2008)