Anonymous Credentials on a Standard Java Card

Patrik Bichsel
IBM Research
pbi@zurich.ibm.com

Jan Camenisch
IBM Research
jca@zurich.ibm.com

Thomas Groß
IBM Research
tgr@zurich.ibm.com

Victor Shoup
New York University
shoup@cs.nyu.edu

ABSTRACT

Secure identity tokens such as *Electronic Identity (eID)* cards are emerging everywhere. At the same time user-centric identity management gains acceptance. *Anonymous credential schemes* are the optimal realization of user-centricity. However, on inexpensive hardware platforms, typically used for eID cards, these schemes could not be made to meet the necessary requirements such as future-proof key lengths and transaction times on the order of 10 seconds. The reasons for this is the need for the hardware platform to be standardized and certified. Therefore an implementation is only possible as a Java Card applet. This results in severe restrictions: little memory (transient and persistent), an 8-bit CPU, and access to hardware acceleration for cryptographic operations only by defined interfaces such as RSA encryption operations.

Still, we present the first practical implementation of an anonymous credential system on a Java Card 2.2.1. We achieve transaction times that are orders of magnitudes faster than those of any prior attempt, while raising the bar in terms of key length and trust model. Our system is the first one to act completely autonomously on card and to maintain its properties in the face of an untrusted terminal. In addition, we provide a formal system specification and share our solution strategies and experiences gained and with the Java Card.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption—*Public key cryptosystems*

General Terms

Algorithms, Design, Performance, Security

Keywords

Anonymous credential systems, Java Card, privacy-enhancing systems, smart card

CCS'09, November 9–13, 2009, Chicago, Illinois, USA.

1. INTRODUCTION

Electronic authentication tokens are spreading rapidly. Applications today already include ticketing, access to buildings, and road tolls. A number of countries have issued electronic ID (eID) cards or are about to do so. All these existing or emerging solutions have in common that the user is fully identifiable in the transactions involving the token. Indeed, many of them offer strong cryptographic identification or qualified digital signatures. The resulting loss of privacy is subject to discussion as pointed out by Huysmans [19], but it is not a severe problem for e-government applications. However, a government-issued root of trust is very attractive for secondary use by (commercial) service providers. Here, privacy becomes a real issue. Indeed, in many commercial applications, unique identification is inappropriate, attribute-based authentication highly desired, and suitable privacy protection essential to make the services sustainable.

For instance, consider a teenager accessing an online chat room by eID. Here, the aim is to restrict access to teenagers only. It is crucial that no data other than the age range of the teenager is revealed to the chat provider. Indeed, if all the eID card's information is revealed and gets into the wrong hands, more damage is done than protection gained. Furthermore, consider a citizen using the eID throughout her entire lifetime and with various third parties. Without sufficient privacy protection, service providers could trace and profile the citizen across organizations. This would lead to an erosion of the citizens' trust and result in the non-sustainability of the entire system. We believe that sustainable secondary use is a is a make-or-break requirement for eID systems as well as for any identity token that supports authentication with third parties.

The ability to build comprehensive user profiles in the context of attribute-based authentication carries the need for strong privacy protection further than mere trust erosion would. It implies the need for full anonymity, which includes unlinkability. Examining identity tokens, and in particular eID cards, over a long time period, then the monotonous growth of identity information at service providers can only be overcome by full anonymity by default. This requirement entails further goals by implication: firstly, we need privacy-enhanced credential systems, namely, anonymous credential systems. Secondly, no (unnecessary) trusted third parties should be involved in transactions, i.e., the credential system must be autonomous. Optimally, the user shall only trust her own identity token and no other principal. Thirdly, if one

considers linkability by timing, the credential system must be able to operate offline, based on long-term certificates.

Let us expand these three thoughts before we analyze the trust model and and hardware setting, in particular, typical smart cards as used to realize eID cards. Luckily, there exist privacy-enhancing technologies addressing our requirement of full anonymity, and allow for attribute-based access control. Anonymous credential systems [22, 9] allow an identity provider to issue an *anonymous credential* to a user. This credential contains attributes such as the user's address or date of birth but also her rights or roles. Using the credential, the user can prove to a third party that she possesses a credential containing a given attribute or role without revealing any other information. For example, in the child-protection example described earlier, the youngster could use a government-issued credential to prove that she fulfills the requirement on the age range. Thus, it seems that what is urgently needed is an implementation of anonymous credentials on tokens, such as smart cards.

The anonymous credential systems proposed by Brands [6] or Camenisch and Lysyanskaya [9] can be implemented on ordinary computers as described in [11] without difficulties. However, it seems they are not suited for implementation on smart cards or USB tokens. Bichsel [3] and Balasch [2] conclude that only systems using joint computation with the terminal can be implemented given the hardware restrictions for the eID scenario. This statement especially holds, if future proof key lengths of at least 1400 bits are considered. But even tremendously reduced systems did not meet the expected transaction times of production eID cards, which are defined to be in the order of 10 seconds.

Beyond the transaction times and key length, there are three more mundane requirements on eID cards imposed by governments and eID technology providers. Firstly, the smart card platform should be standardized, for governments and eID technology providers shy away from proprietary technology lock-ins.[1] Also, we envision the anonymous credential system to be deployed as a complement to existing eID systems and not to replace other authentication mechanisms. Even though one could achieve much more efficient solutions with a native card implementation, this would severely hamper the acceptance for the proposal. We therefore base our work the Java Card 2.2.1 standard [26].

Secondly, the eID card must be certified, for instance in a Common Criteria for Information Technology Security Evaluation [14]. Clearly, we need to aim at making the certification gap as small as possible and, therefore, use an off-the-shelf smart card with comprehensive certification.

Thirdly, the smart card platform must be well-established and cheap. We therefore restrict ourselves to smart cards that are 3–4 years old and in production in current eID systems. We also follow the standard operation procedures of these smart cards, e.g., to avoid write-operations to EEPROM whenever possible.

The main obstacle to implementing anonymous credential systems on such cards seems to be twofold. Firstly, we need to execute fast modular exponentiations, which requires the use of the card's cryptographic co-processor. However, the interfaces offered by the (off-the-shelf) cards' operating system do not give direct access to this, but only offer high-level functionality such as RSA encryption. Consequently, we will use the limited interfaces that standard Java Cards provide. Secondly, typical smart cards are rather limited in the amount of RAM that can be used for computations. This makes it, for instance, hard to store all intermediary results during an authentication transaction.

[1] Of course, this requirement also transfers to the standardization of anonymous credential systems.

1.1 Related Work

There have even been several approaches to implement anonymous credential systems on smart cards. Bichsel [3] and Balasch [2] focus on providing the arithmetic functionality required by anonymous credential systems, i.e., fast modular arithmetic. Balasch implements the arithmetic using AVR microcontrollers, whereas Bichsel uses the JCOP platform. Danes [16] provides an analysis of different trust models and compares them with respect to security and privacy. He projects computation times using the hardware specification, implicitly assuming a custom operating system, and obtains an execution time of 6 seconds for his preferred protocol. This protocol still assumes trust in the terminal. We provide a comparison of the measurements of the three approaches with ours in Table 1. Note that the table focuses on the computation times of the core anonymous credential system and does not account for additional computations such as revocation equations.

Given that the authors use very different systems, we want to analyze the systems on the basis of single exponentiations. The difference to Bichsel [3] is apparent and does not need further explanation. The implementation by Balasch [2] can be compared by extrapolation of his measurements. An exponentiation of a base/modulus bit length of 1984 with an exponent of 1024 bits accounts to roughly 270 seconds. We are able to compute such an exponentiation in 1.3 seconds.

1.2 Our Contributions

In this paper, we overcome the technical limitations to implement the Camenisch-Lysyanskaya (CL) anonymous credential system on a standard Java Card. We do this by exploiting the RSA encryption interface in a number of ways and by clever management of the available resources (especially RAM). In fact, our implementation can execute a proof of possession of a credential in a few seconds, which is fast enough for a multitude of eID use cases. Thus, we believe to have overcome the possibly final *technical* barrier for privacy-protecting electronic identity tokens.

Our contributions are twofold. Firstly, we discuss the challenges of actually implementing the CL credential system on a Java Card. In particular, we consider the severe platform restrictions, which entails concise analysis of the available interfaces as well as careful treatment of the hardware resources. In addition, we share our experiments, experiences, and strategies to overcome these limitations. Our solutions enabled us to outperform all prior anonymous credential system proposals on smart cards by several orders of magnitude. Moreover, our insights and tricks can be of merit for other implementations of advanced cryptographic primitives on Java Cards.

Secondly, we report the first practical implementation of an anonymous credential system on a standardized, off-the-shelf Java Card, a JCOP v2.2/41. We use a variant of the standardized Direct Anonymous Attestation (DAA) protocol [7], and demonstrate the feasibility of such a system for actual eID cards. In contrast to prior proposals, our smart card credential system is autonomous, that is, it forgoes any

Table 1: Overview of different approaches to establish anonymous credential systems on a smart card. We compare the implementations in terms of the transaction time, even though prior systems only execute a partial proof and use smaller key length. We denote the system parameters with base bit length ℓ_b, modulus bit length ℓ_n and maximal exponent bit length of ℓ_e by $\ell_b{}^{\ell_e}(\ell_n)$.

	Danes [16]	**Bichsel** [3]	**Balasch** [2]	**This Paper**		
Date	2007	2007	2008	2009		
Bit Length	—	$72^{344}(72)$	$1024^{1752}(1024)$	$1280^{735}(1280)$	$1536^{895}(1536)$	$1984^{1152}(1984)$
Transaction Time	—	450s	133.5s	7.4s	10.5s	16.5s
Trust Model	trust terminal	trust terminal	trust terminal		autonomous	
Implementation	none, prediction of transaction time	on Java Card JCOP v2.2/41	AVR 8-Bit RISC microcontroller		on Java Card JCOP v2.2/41	

joint computation with the terminal. Our system not only guarantees the secrecy of the user's master key during the card's complete life cycle, but also protects user's privacy in face of an untrusted terminal. Our system goes far beyond a pure demonstration as it achieves production-quality parameters for eID cards. This includes strong key length of 1536 bits for the strong RSA modulus, transaction times on the order of seconds, and very modest hardware requirements (see Table 1). In fact, it could be applied to eID Java Cards currently being rolled out in various European countries.

1.3 Outline

The remainder of the paper is structured as follows. We begin with a discussion of the requirements in Section 2, where we start with requirements that are imposed by the eID scenario and continue with functional requirements that rise in the context of secondary use of eID cards. Section 3 elaborates on the underlying cryptographic system, design decisions and concludes with the protocol specification. In Section 4, we illustrate the main obstacles we encountered when realizing the system we specified, the solutions we developed, the architecture we built, and the measurements we performed. We discuss the results of our implementation with respect to the requirements in Section 5 and conclude with Section 6, where we provide an outlook on future development of anonymous credential systems on smart cards.

2. REQUIREMENTS

We base our requirements discussion on the scenario of eID cards for three reasons. Firstly, eID technology is likely to pervade entire societies and to affect the life of many citizens. Secondly, its actual hardware platform is particularly challenging for implementing an anonymous credential system. Thirdly, it allows us to intuitively motivate requirements that abstractly hold for any application involving personal tokens with severe resource restrictions.

2.1 Application Requirements

Let us begin with the requirements dictated when using an eID card for applications having non-government organisations as service providers.

Sustainable Secondary Use. The users must be able to use their eID card over their entire lifetime without privacy or trust degradation. A continuous strong privacy protection for all transactions is crucial. This is a key requirement that we are going to meet by using an anonymous credential system.

Autonomous Trust Root. A wide range of trust scenarios must be supported without drawbacks on security or privacy. Particularly, the card must act securely in face of an untrusted or malicious terminal[2]. Therefore, the anonymous credential system must protect the citizen's security and privacy autonomously and cannot (easily) delegate computations to the terminal.

The privacy discussion gains in complexity with the introduction of variable attribute policies, as the terminal may attempt to send the eID card multiple policy requests—without the citizen's knowledge or consent—to infer a profile of the citizen. As the eID card is in principle stateless, it is at the mercy of the terminal. The terminal can easily reset the card and send another policy request. Naive solutions to store the card's state or create an audit log of the terminal's requests are not easily feasible because of the cards limitations in write/erase cycles on persistent memory. Proposals that certify card readers as well as applicable policies are used to confine a potential exposure, be it in terms of obtainable attribute set or of potentially malicious readers.[3]

Long-term Certificates. An eID card must forgo short-term updates, particularly of the keys and certificates, be it because some countries support offline[4] applications (such as vending machines), or because some countries ban card updates outside of a trusted environment. In privacy terms, this allows to prevent a linking by timing.

Performance. An anonymous credential system for eID cards faces stiff performance requirements, notably, the need to complete transactions in mere seconds.

Future Proof Key Length. Currently, lengths of an SRSA modulus size greater than or equal to 1400 bits are considered future proof.

2.2 Functional Requirements

Clearly, unique identification, qualified signatures, and disclosure of the citizen's full address are important func-

[2]From a user perspective, sharing data with the own device has different implication compared to sharing with a third-party terminal, e.g., at a bar, or an Internet cafe.

[3]Our system can easily realize a check of the terminal's attribute policy and restrict the disclosable attributes for uncertified terminals. However, these proposals do not constitute a real solution of the problem at hand, and further research is required in this area.

[4]Offline, here, refers to the terminal being able to serve the request of the card without an online connection to the authorities or to an identity provider.

tional requirements for eID cards; however, we focus on functional requirements with stronger privacy properties.

Proof of Possession. The card must be able to issue a proof of possession of a credential. Thus, proving the value of an attribute without leaking any information about the attribute value.

Age proof. Nowadays, eID cards are often used as basis for a proof of age, mostly in the area of youth protection. Contrary to the common perception of an age proof as a means to show adulthood and to obtain restricted goods (medias, alcohol, cigarettes), age proofs are also important to establish protection zones for youngsters on the Internet.

Finite-set Attributes. Also, eID cards contain a variety of binary or finite-set attributes that are particularly privacy-sensitive [8]. Consider, for instance, attributes of health and special status: visually or hearing impaired, social benefit recipient, unemployed, or elderly. Undoubtedly, these attributes need to be disclosed only selectively, or even only issue a proof certifying that the citizen is entitled to receive social welfare by holding one out of many attributes.

Revocation. Revocation is of central importance for eID systems. The card needs to be revoked when the owner declares her eID card lost or stolen. As the traditional approach of revocation lists implies privacy hazards for honest citizens, we need to explore privacy-preserving revocation mechanisms.

2.3 Hardware Requirements

Let us summarize our hardware challenge: Our goal is to establish an autonomous credential system on a smart card with the following properties: (i) a standardized Java Card with comprehensive security certification, (ii) used by existing eID systems in production, and (iii) with restricted write/erase-cycles. We use the Java Card 2.2.1 standard [26] interface, which prevents direct access to the cryptographic co-processor, fast multiplication, and exponentiation primitives. It only offers the use of well-defined primitives such as RSA encryption. In addition, transient memory is severely restricted (750 bytes heap, 200 bytes stack), which makes the implementation of multi-base exponentiation and many pre-computation techniques virtually impossible.

These severe limitations explain why prior proposals [3, 2] could only achieve transaction times on the order of minutes, despite the fact that they delegated most computations to the terminal.

3. PROTOCOL DESIGN

Let us consider the protocol design for a standard Java Card in stages. Firstly, we review the cryptographic variants of anonymous credential systems. Secondly, we discuss the options for hardware trust. Thirdly, we present out design decisions that follow these arguments.

3.1 Cryptographic Alternatives

Anonymous credential systems were introduced by Chaum in [12, 13] and subsequently improved, in particular, by Brands [6] as well as Camenisch and Lysyanskaya [9, 10]. Relations based on blind signatures such as those by Brands have a severe drawback when it comes to implementations on a smart card: Proving possession of a credential in an unlinkable, i.e., privacy-maintaining, way requires the issuance of a new credential, which would exhaust the EEPROM write cycles quickly. Identity mixer, developed by Camenisch et al. [21], does not suffer from this limitation, i.e., one credential can be used repeatedly to prove its possession without these proofs becoming linkable.

Therefore, we have chosen the Camenisch-Lysyanskaya (CL) [9] signature scheme as basis for our lightweight credential system on Java Card. Let us first consider the variants of Camenisch-Lysyanskaya itself: The most common one is based on the Strong RSA assumption and specified in the Identity Mixer protocol suite [21]. Subsequently, Camenisch and Lysyanskaya proposed alternatives based on bilinear maps that rely on the LRSW assumption [9], and one that build on the Boneh-Boyen-Shacham group signature scheme [4]. The latter was improved upon by Au, Susilo, and Mu [1]. The bilinear map variants of the CL signature scheme can operate in smaller prime-order groups, whereas the SRSA variant requires a large composite modulus. Thus, the bilinear map variant is advantageous in general, particularly as the SRSA variant has the client operate with unknown group order. Nevertheless, we dismiss the bilinear maps based variant as the smart cards considered do not offer suitable algebraic support for the required elliptic curves.

3.2 Hardware Resilience

We need to consider an important balance question for eID cards: To what extent can we trust the hardware's resilience and how much do we need to rely on cryptographic protection? We note that typical eID cards are tamper-resistant and equipped to protect their private keys for identification and qualified signatures. Thus, we can assume that it is costly to break/clone a single eID card, and that an attack of the tamper-resistance of one card does not easily transfer to attacks of other cards. Thus, the damage is local as otherwise the system of eID cards would be broken as a whole.

As a means of mitigating the damage of broken or stolen cards, an eID system needs provisions for revocation. Typically the issuing authority would be in charge of revoking cards once a local breach has been detected. The eID scenario holds more potential impact associated with breaking the tamper-resistance of identification and qualified signatures than the attribute-based authentication. Therefore, it is, in principle, sufficient to have the same protection standards as for the other pillars of eID functions and, by extension, good enough to trust the hardware resilience for our use cases.

Finally, we conclude that the resilience of eID cards is an important protection feature that mitigates potential breaches. Under the condition of a sufficient revocation system, it is possible to choose more efficient cryptographic mechanisms while maintaining the same level of protection.

3.3 Design Decisions

Implementing the full-fledged CL anonymous credential system would not be feasible on current cards. That is, features such as an age-proof (i.e., proving that the date of birth contained in the credential issued has a distance of at most n years from the current date) or encoding all the more than 20 typical fields of a standard identity card and allowing selective disclosure for each of them would result in a computation time on the order of 70–100 seconds. As this would not be suited for practice, we rely on the tamper resistance of the hardware for such attribute-related proofs.

Thus, similarly to the model the Trusted Computing Group has taken for their TPM chips with the Direct Anonymous Attestation (DAA) protocol [7], we have a two-stage approach. We use anonymous credentials to have the smart card prove that it is a valid (and intact) card and therefore can be trusted to make statements about its bearer. These statements, e.g., an age proof, are then made by the card itself. Consequently, the correctness of these statements is not enforced cryptographically but by the tamper resistance of the card. Thus, we will have to protect ourselves against the case when a card is broken open and the cryptographic credentials are extracted. In this case, the extracted credential can be used to back any attribute-related statement. However, breaking a card is costly and will mostly be done for economical reasons. Thus, employing techniques that protect from massive sharing might be the most appropriate action.

Our proposed solution is therefore as follows. We issue a Camenisch-Lysyanskaya credential [9] with a secret key m_0 on an eID card and store all attribute information about the citizen in the card independently of the credential.[5] When the citizen wants to use the cards for some privacy-protecting authentication, we let the eID card compute a valid attribute statement (based on the attribute information stored on the card) and sign it with the Fiat-Shamir heuristic [17] during a proof of possession of the issued credential. On top of that, the card provides a discrete log commitment, i.e., $C = g_r^{m_0}$ on the secret key m_0 with a random base g_r during each transaction (where it is ensured by the proof of possession that this is chosen correctly).[6] This commitment is a pseudo-random value with computational hiding properties. However, it allows for the detection of revoked cards as authorities can check C against $g_r^{\hat{m}_0}$ for each $\hat{m}_0$ retrieved from the revocation list and, if there is a match, decline the transaction (or take legal action).

3.4 Protocol Specification

The system setup starts with the initialization of the smart card, which can only be executed once. It continues with the issuance of at least one credential. From that point on, a proof of possession can be executed. Additional credentials can be issued and bound to the card at any point later on.

Smart Card Setup. The master secret m_0 can only be set once for each card with $m_0 \in \{0,1\}^{\ell_m}$. It will be used to bind all certificates issued to a card together and to the card. It is generated by the card and released only computationally hidden.

After the setup of the smart card, a credential is issued to it. During this process the issuer public key comprising a modulus $n = pq$, where p, q are safe primes that fulfill $p = 2p' + 1$ and $q = 2q' + 1$, and p', q' are primes, is needed. This key also contains bases $Z, S_2, R_0, \ldots, R_i \in_R \langle S_1 \rangle$ where S_1 is an arbitrarily chosen quadratic residue modulo n and $\langle S_1 \rangle$ denotes the group generated by S_1. A more detailed description of the issuer key generation can be found in [21]. We chose the relevant bit lengths as follows: $\ell_n = 1536$, $\ell_m = 256$, $\ell_e = 592$, $\ell'_e = 120$, $\ell_v = 768$, $\ell_\varphi = 80$ and $\ell_{\mathcal{H}} = 160$. In general, ℓ_k denotes the bit length of parameter k. The bit lengths ℓ'_e, ℓ_φ and $\ell_{\mathcal{H}}$ define the length e', the bit length used to achieve statistical zero knowledge, and the bit length of the hash values, respectively. Note that the parameter ℓ_v is much shorter than suggested in the Identity Mixer specification [21] because two blinding bases are used.

After having run the issuance protocol successfully, the smart card holds a valid CL signature (A, v, e). We want to discuss the proof protocol. The issuance protocol entails similar challenges and benefits from the same solution strategies. We leave the details to the full paper.

Proof of Possession Protocol. Proving possession of a certificate follows the lines of argumentation of the Identity Mixer protocol. As the card cannot handle exponents that are larger than the modulus, we split the long exponents into two shorter ones at the cost of computing an extra exponentiation. More precisely, instead of computing S^v we compute $S_1^{v_1} S_2^{v_2}$ with $S_1 = S$, $S_2 = S^{2^\ell}$ and $v = v_1 + v_2 2^\ell$ for a suitable ℓ. The verifier starts the protocol by sending a nonce $n_1 \in_R \{0,1\}^{\ell_{\mathcal{H}}}$ to the prover, which guarantees the freshness of the proof. The prover continues by first choosing $v_1^*, v_2^*, r_{g_R} \in_R \{0,1\}^{\ell_v + \ell_\varphi}$ and subsequently computing the following values.

$$A' := A S_1^{v_1 *} S_2^{v_2 *} \pmod n \qquad g_R := S_1^{r_{g_R}} \pmod n$$
$$\bar{v}_i := v_i - v_i^* e,\ i \in \{1,2\} \qquad C := g_R^{m_0} \pmod n$$
$$e' := e - 2^{\ell_e - 1}$$

The card sends $A', (g_R, C)$ to the terminal, which forwards it to the verifier. In addition, the card calculates $\hat{e}, \hat{m}_0, \hat{v}_1, \hat{v}_2$ and the hash c, and sends those values to the terminal. For the calculation mentioned, the card chooses $\tilde{e} \in_R \pm\{0,1\}^{\ell'_e + \ell_{\mathcal{H}} + \ell_\varphi}$, $\tilde{m}_0 \in_R \pm\{0,1\}^{\ell_m + \ell_{\mathcal{H}} + \ell_\varphi + 1}$ and $\tilde{v}_1, \tilde{v}_2 \in_R \pm\{0,1\}^{\ell_v + \ell_{\mathcal{H}} + \ell_\varphi}$ at random. To continue with the calculation of the proof, the following values are computed:

$$\tilde{T} := A'^{\tilde{e}} R_0^{\tilde{m}_0} S_1^{\tilde{\bar{v}}_1} S_2^{\tilde{\bar{v}}_2} \pmod n \qquad \hat{e} := \tilde{e} + ce'$$
$$\tilde{C} := g_R^{\tilde{m}_0} \pmod n \qquad \hat{m}_0 := \tilde{m}_0 + cm_0$$
$$c := \mathcal{H}\left(sysparam, \tilde{T}, \tilde{C}, n_1\right) \qquad \hat{v}_i := \tilde{\bar{v}}_i + c\bar{v}_i,\ i \in \{1,2\}$$

The verifier can check that the smart card possesses a valid credential by computing the challenge $\hat{c}$ and comparing it with the submitted challenge c.

$$\hat{T} := \left(\frac{Z}{A'^{2^{\ell_e - 1}}}\right)^{-c} A'^{\hat{e}} R_0^{\hat{m}_0} S_1^{\hat{v}_1} S_2^{\hat{v}_2} \pmod n$$
$$\hat{C} := C^{-c} g_R^{\hat{m}_0} \pmod n$$
$$\hat{c} := \mathcal{H}\left(sysparam, \hat{T}, \hat{C}, n_1\right)$$

[5] Our implementation is capable of including more attributes in the credential as well as handling them in zero-knowledge proofs of knowledge and selective disclosure. Each additional attribute exponent comes at a cost of 1684ms transaction time at a modulus bit length of 1536 bits. This accounts for the modular exponentiation, required multiplications, additions and PRNG calls. Of this, 1016ms are pre-computation, 668ms are policy-dependent. We have tested this functionality, yet do not propose it as primary solution.

[6] This discrete log commitment needs to be computed separately from the performance measurements we provide in Table 3. The card needs to generate a random base and compute the commitment as well as prove its representation in zero-knowledge. This costs several calls to the PRNG to generate 1536 bits for a pseudo-random base and two ModExp. The response for the zero-knowledge proof of m_0 can be reused. With a 1536-bit modulus, this makes an additional transaction time of 1474ms, which can be fully handled at pre-computation time.

Subsequently the lengths of $\hat{m}_0$ and $\hat{e}$ have to be verified with $\hat{m}_0 \in \{0,1\}^{\ell_m+\ell_\varphi+\ell_\mathcal{H}+1}$ and $\hat{e} \in \{0,1\}^{\ell'_e+\ell_\varphi+\ell_\mathcal{H}+1}$. It is straightforward to verify that $\tilde{T}$ equals $\hat{T}$.

Finally, checking whether the certificate has been revoked is done as follows. Assume that $(m_{(0,1)}, \ldots, m_{(0,f)})$ for some f is the list of revoked secret keys (i.e., the list of the secret keys that have been extracted from tokens). For each $m_{(0,j)}$ check that $C \neq g_R^{m_{(0,j)}} \pmod{n}$.

This proof protocol does not disclose any attributes and thus implements the DAA case. The extension of adding either disclosed or hidden attributes is straightforward [21].

4. REALIZATION ON A SMART CARD

Given the cryptographic design decisions and the formal system specification, we now elaborate on the realization on an actual off-the-shelf Java Card in four steps: firstly, an analysis of the JCOP environment, secondly, strategies that can partially overcome the limitations, thirdly, integration of these aspects in a sketch of our high-level system design, and, finally, a report of the performance achieved.

Our system requires modular multi-base exponentiation, multiplication, and addition, all with a large composite modulus and without being privy of the group order. Furthermore, we need random numbers, digests for Fiat-Shamir, and a cache for intermediary results. We analyze the obstacles presented by the card, and derive optimizations methods. We achieve much by tunneling computations to the card's hardware accelerator and reducing other operations to the accelerated ones. Unfortunately, this hardware accelerator is well encapsulated behind the Java Card's high-level crypto interface, so that we resort to disguising credential system computations as RSA encryption operations.

In the following, we discuss the limitations of a Java Card, be it in terms of interfaces or be it in implementation environment (e.g., available RAM). We then show how our lightweight credential system can nevertheless be implemented, highlight key architecture concepts, and conclude with a discussion of the performance of our implementation. This last part shows that privacy-protection tokens are practical today.

4.1 JCOP Environment

The Java Card 2.2.1 standard [26] offers a well-defined set of interfaces to implement custom applications. We used a standard-compliant JCOP smart card [20], which imposes further limitations when it comes to low-level operations. We discuss the interfaces that are most relevant for our implementation. We start with basic restrictions such as RAM and 8-bit arithmetic, and continue with cryptographic primitives.

RAM Restrictions. On top of the restricted access to its crypto acceleration, a smart card has scarce transient memory. Our JCOP v2.2/41 card is equipped with 2304 bytes of RAM. This transient memory is distributed among the Java™stack, APDU buffer (used for communication between card and environment), atomic transaction buffer, and Java heap. For the calculations we can only make use of the transient heap, which is 750 bytes.

EVALUATION 1. *As we aim at reasonable modulus sizes of at least* 1400 *bit, each group element already requires at least* 175 *bytes of transient memory. Moreover, to compute the zero-knowledge proofs as specified in Section 3.4, we need to juggle multiple group elements in RAM at the same time. Therefore, transient memory is a highly limiting factor.*

8-bit Arithmetic. The JCOP v2.2/41 Java Card comes with an 8-bit processor/ALU. All arithmetic operations such as addition, subtraction, and multiplication are delegated to it. Whereas the built-in arithmetic operates on byte and short values, we require operations of arbitrary-length integers. This is either supported by BigInteger libraries in newer smart cards or custom-implemented in the application layer. In any case, it is very costly. Our particular hardware contains a FAME-X (Fast Accelerator for Modular Exponentiation - Extended) crypto co-processor that features support for modular exponentiations. It is not directly accessible from the application layer of Java Cards.

EVALUATION 2. *Any attempt to have the exponentiations or multiplications computed by the* 8*-bit ALU is bound to fail, and will result in transaction times as highlighted by Bichsel [3] for Java Cards and Balasch [2] for AVR microcontrollers. Our best result for a pure application layer implementation of a* 248*-byte addition was* 76*ms. Projecting resulting exponentiation times indicates that we need to do better than that.*

Random Number Generation. The JCOP v2.2/41 smart card offers true random number generation (TRNG) and pseudo random number generation (PRNG). Note that the PRNG expects a strong random seed and infuses further randomness sources, i.e., a standard-compliant PRNG does not produce the same outputs deterministically if seeded with the same number.

EVALUATION 3. *The proofs specified in Section 3.4 require the generation and reuse of multiple random exponents. The severe RAM limitations deny us the option to store the randomness and the non-deterministic behavior of the PRNG denies its re-computation using the provided functionality.*

SHA-1 Interface. The SHA-1 interface allows the hashing of messages of up to $2^{64} - 1$ bit length to a 160-bit string. SHA-1 is implemented in software on the JCOP v.2.2/41 that we use, and therefore, relatively slow. In addition, digest updates have to respect the block size of 64 bytes, which makes certain key lengths, i.e., 1984 bits, less favourable as we would need to hash 2048 bits.

EVALUATION 4. *The SHA-1 primitive is only a second-rate candidate to generate and recompute pseudo-randomness. We shy away from its slow software implementation and the transient memory impact.*

DES/3DES Interface. The symmetric encryption interface offers a variety of modes, be it in terms DES, 2 key 3DES or 3 key 3DES, in terms of cipher block chaining mode (CFB) or electronic codebook mode (ECB), or in terms of the padding scheme.

EVALUATION 5. *For us, it is of particular importance that the JCOP v2.2/41 card offers hardware acceleration for 3DES operations and that there exist efficient and secure pseudo-random number generators based on 3DES.*

DSA Interface. The DSA signing primitive uses various exponentiations that might be leveraged for our purposes. In particular, it executes a multi-base exponentiation with configurable bases.

EVALUATION 6. *The DSA key interface allows us to specify the public key and private key, but not the value of the exponents. We perceive the involvement of the hash function as obstacle to using the DSA interface for our intended acceleration of arithmetic operations.*

RSA Interface. The Java Card 2.2.1 standard [26] offers RSA [25] encryption and decryption, either in normal mode or with Chinese Remainder Theorem (CRT) support for private key operations with known factorization. The RSA public key consists of the modulus n and the public exponent e, whereas the private key contains the secret exponent $d = e^{-1} \pmod{(p-1)(q-1)}$ and modulus factorization $n = p \cdot q$. The parameters p and q are chosen as random prime numbers that are of similar length and not equal. A message m is encrypted to $c = m^e \pmod{n}$ using the public key. The decryption uses the private key and retrieves $m' = c^d \pmod{n}$.

For the Java Card 2.2.1 standard, the public exponent e is usually quite small (4 bytes) and often fixed to common exponents such as 3 or Fermat-4. Whereas the JCOP environment allows us to set exponents and moduli of the RSA keys in a wider value range, it still limits the bit length of exponent (ℓ_e) and base (ℓ_b) to at most equal the bit length of the modulus ℓ_n.

Moreover, the Java Card 2.2.1 standard only allows computations on persistent keys (EEPROM), as normal RSA encryption operates on long-term keys.

EVALUATION 7. *In principle, the RSA primitive sounds like a good candidate to tunnel computations for the credential system to the hardware acceleration. We face three limitations: firstly, the constraint to small standard exponents for encryption may foil our endeavor altogether. Secondly, the limitation to modulus-size exponents conflicts with the blinding of the credential system: it must be larger than the modulus to stay provably secure. Thirdly, the restriction to persistent keys bars us from exploiting the interface directly: we need to update the keys frequently to obtain exponentiations with random values and would therefore cause many write cycles—slow death—to the EEPROM.*

Summary. We have seen that, firstly, a Java Card—and in particular the JCOP v2.2/41 card—imposes severe limitations in terms of transient memory and 8-bit arithmetic on credential system operations, such that a delegation to hardware acceleration is unavoidable. Secondly, the exposed cryptographic interfaces are well encapsulated, either completely unusable for our endeavor or posing further technical obstacles. Thirdly, the RSA interface is promising, but requires a new implementation of transient keys with long public exponents to serve our purpose. Also, it would conflict with compliance with the Java Card 2.2.1 standard.

4.2 Our Solution Strategies

In general, one can solve most of the restrictions indicated using the computation time versus storage trade-off. Balasch shows in [2] some results in this direction. However, we could not allow ourselves this luxury: given the very tight bounds on all relevant metrics with our Java Card, we had to look for other solution strategies.

Multi-base Exponentiations. Undoubtedly, multi-base exponentiations are the most important operation in our protocol. The combination of not having an interface to exploit hardware-accelerated multi-base exponentiation, and computing multi-base exponentiations in the application layer consuming too much transient memory, we dismissed this option altogether. Also, it falls back on a custom implementation on the 8-bit ALU that is too slow. We resort to hardware-accelerated modular exponentiations.

Modular Exponentiation. Modular exponentiation that is implemented on the application layer exhibits a devastating performance, which when holds when using advanced methods such as Montgomery reduction.

IDEA 1. *We delegate modular exponentiations to the RSA encryption and overcome interface limitations (Section 4.1) as follows:*

- *By creating a new transient RSA key design that supports public exponents in modulus length and a rapid change of exponents in transient memory. This is made possible by the use of special library and violates the Java Card 2.2.1 standard. Note, that the Java Card 3.0 standard does allow RSA keys to be stored in transient memory.*
- *By modifying the credential system to execute the blinding over two bases S_1 and S_2 instead of a single base S, thus maintaining provable security with smaller exponent sizes.*

Let us consider this in detail: firstly, RSA keys normally reside in EEPROM, or even worse, in a protected EEPROM section. Therefore, executing many exponentiations with changing RSA keys, will deplete the write cycles of this particular EEPROM section very quickly. In addition, writing to EEPROM takes much longer than writing to RAM[7]. We overcome this limitation by creating a new RSA key structure that resides in transient memory. Although Java Card 2.2.1 does not support RSA keys in transient memory, JCOP actually does, and we exploit this in our implementation. Note, however, that the newer standard Java Card 3.0 does support RSA keys in transient memory, and so, looking forward, our solution will be standard-compliant.

Secondly, we overcome the exponent and base length limitations, which prevented us from carrying out the Identity Mixer computations as defined in [21]. Specifically the exponent for blinding the certificate needs to be larger than the modulus. The solution to this problem is the use of two bases with two independently chosen exponents which results in the equation given in Section 3.4.

Modular Multiplication. Modular multiplication is the second most important primitive when it comes to implementing the Identity Mixer anonymous credential system. It is also too heavy-weight for the application layer. Luckily, we succeeded in building an extremely efficient modular squaring primitive that overcomes this issue.[8]

IDEA 2. *We reduce multiplications to highly efficient squaring operations on the hardware acceleration by employing a* binomial formula*. In particular, we compute the modular multiplication of a and b modulo n by computing $((a+b)^2 - a^2 - b^2)/2 \pmod{n} = ab \pmod{n}$.*

Because of the small exponent, the computation is very efficient. The subtraction is implemented naive, which makes

[7]Writing a page (1-64 bytes) to EEPROM typically takes 1.6ms according to [23].

[8]A modular squaring of a 1984-bit number with the hardware acceleration takes 9ms.

it the predominant factor when it comes to computation time. The final division translates to a simple shift operation. Using the optimizations outlined, we can reduce the computation complexity in the application layer from $O(\ell_n^2)$ to $O(\ell_n)$, where ℓ_n is the length of the modulus.

Addition. Given our optimizations of exponentiation and multiplication, the addition and subtraction become predominant when it comes to performance. As production cards can be easily patched to expose a fast byte-array addition primitive, we base our smart card implementation on an application-layer arbitrary position integer addition. However, the following optimization can lead to a considerable protocol speed-up even with a standard card.

IDEA 3. *We could delegate the addition to the hardware acceleration by tunneling it through the RSA-CRT decryption operation. By carefully setting the base and exponent arguments the CRT algorithm produces an addition/subtraction in the decrypted message that can be extracted by inexpensive shifts. We refer to the full paper for details on this method.*

Randomness. The Java Card offers a true random number generator. However, we cannot store the randomness for the proofs because of the severe memory limitations (see Section 4.1). We therefore need to regenerate pseudo-random values on demand. As the Java Card 2.2.1 standard specifies that the pseudo-random generation with the same seed will result in the same random number, we need an alternative mechanism.

IDEA 4. *We create our own pseudo-random number generator that allows us to regenerate randomness identified by variable names. We generate the seed with the true randomness generation of the JCOP card and use the formal state machine of Section 4.3 to enforce that a each proof is executed with a fresh random seed.*

In the current implementation, we use the SHA-1 hash function to generate pseudo-randomness, however, this leaves much room for optimization: using the dedicated 3DES coprocessor as PRNG would enable an additional performance gain. Considering the computation times of 3DES, which are specified as $< 35\mu$s [20], and comparing to the measurements in [2] (3ms per SHA-1 Op) as well as our experiments (22ms per 100-byte PRNG data), it is safe to estimate the benefit of this measure to roughly half the computation time of the pseudo randomness.

Transient Memory. We mitigate the scarce resources problem of transient memory by partially using memory dedicated to a fixed component. In our example, we use the card's communication buffer. With a length of 255 bytes, it has a reasonable size to be exploited. This approach carries the major risk that the buffer might be read or changed by other applets. Thus, we need to make sure that no sensitive data resides in this memory.

IDEA 5. *We use the communication (APDU) buffer of the smart card as additional transient memory. For security reasons we enforce that any data written to buffer is non-sensitive or already cryptographically blinded. This means in particular that a proof's randomness, the user's master key, and the attribute values are never written to the APDU buffer.*

Summary. We created a toolbox for efficient computation of various algorithmic components of credential systems. It helped us to make the most of our situation, in view of its high expectations (future-proof key length and short transaction times) and severe limitations (RAM, 8-arithmetic, limited crypto interfaces). In particular, it enabled us to create the credential system on card as specified in Section 3.4 with the architecture and performance as described below.

4.3 Architecture of the Full System

Whereas we dedicated the previous sections to overcoming the low-level obstacles of the JCOP environment, we now take a step back to present the high-level architecture of the overall system. After all, realizing a full-fledged anonymous credential system on a Java Card is not just algorithms and tricks to achieve fast exponentiations, but requires serious consideration at the system level.

Our main requirements on the architecture are twofold. Firstly, it must strongly economize the Java Card's resources, and in particular, use transient memory optimally and in a tightly controlled manner. Secondly, it must feature strong security and robustness properties, i.e., justify its use in high-trust areas such as eID. We briefly discuss these two requirements by mentioning the core points of our architecture and complementing them with a design overview.

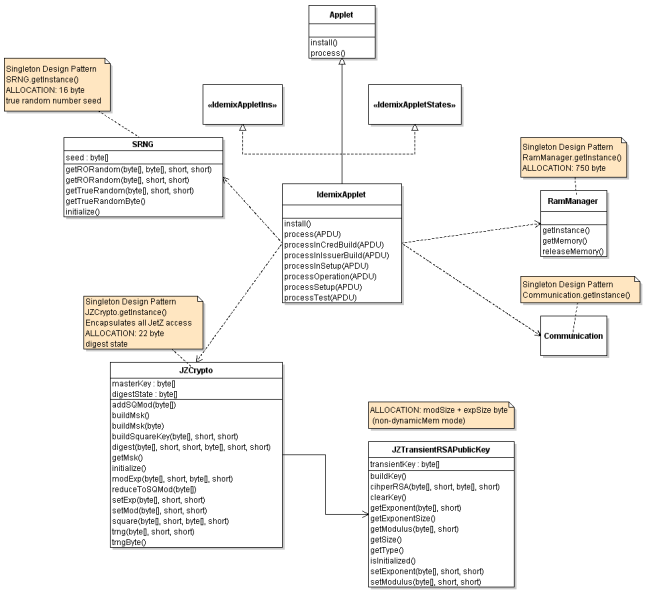

Figure 1: Overview of the class design of the credential system for Java Card.

Let us start with the economy aspects, which we complement with the class design overview of Figure 1. Transient memory clearly is the sparsest resource of the card, particularly because we juggle multiple large byte arrays with group elements. We therefore first established an explicit RamManager that owns most of the applet's memory. It governs byte arrays for group elements as well as exponents and organizes the request and release of this memory. Secondly, we created most classes either in the *singleton* design pattern [18] or as *static*, such that there exists either only one instance with state or that the class does not have dynamic state at all. This design is not only for economy but also includes security features in terms of information flow/non-

interference: the RamManager, for instance, guarantees that byte arrays are zeroed before reuse. Also, security-critical memory, such as digest state and random seed for PRNG, is completely separated from other computations and well encapsulated in the corresponding classes.

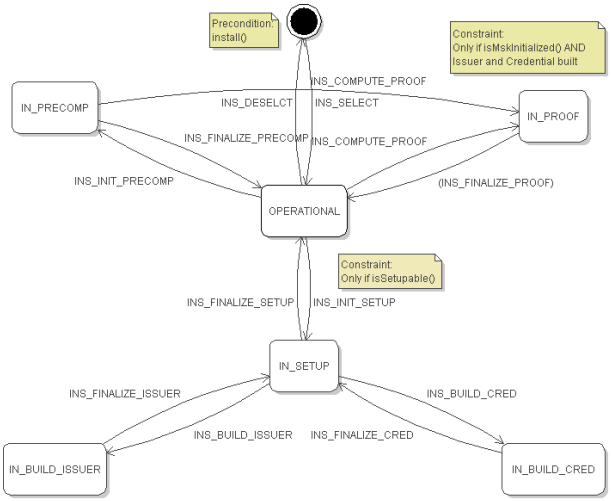

Figure 2: State machine of the anonymous credential system applet.

The security properties of the applet go beyond information flow control and, in particular, ensure consistency of the card's state. This can be on an atomic transaction level or on the system-state level. We solve the first part mostly through *Factory* design patterns [18][9] and prudence for all write operations to the card's EEPROM, which we realize with the Java Card's atomic transaction facility[10] and consistency checks before committing transactions. We solve the latter part with a formal state machine, depicted in Figure 2. It establishes tight control on setup and operational states, acceptable inputs, and potential transitions. Even though we thoroughly tested the applet with white and black box tests, we separated out all test functionality, which are interfaced by the abstract method processTest() and not compiled into the production version (similar to the *Visitor* design pattern [18]).

4.4 Performance

We performed measurements executed on a JCOP v2.2/41 smart card. We first measured the performance of the arithmetic operations. Especially, the modular exponentiation is of interest to us. Running 500 consecutive executions of an exponentiation using a base and a modulus with bit length of 1984 bits and an exponent of length 1024 bit, we measured a computation time of 1.3 seconds for each exponentiation. Cutting the exponent in half reduces the computation time by a factor of 2. Furthermore, the computation time of squaring a 1984-bit base using a modulus of the same length results in a computation time of approximately 15ms.

Our main interest lies in the computation times of the protocol proving holdership of a credential as described in Section 3.4. Note that the exponentiation for the revocation as specified is not included in the measurements. The computations of the credential issuance are less important as there are only a few credentials issued to a card, but possibly a large number of proofs of possession. Also, computations of the credential recipient and an entity proving possession of a credential are very similar, and timings can be well approximated.

We analyzed the performance using different key lengths starting with a 1280-bit modulus up to a modulus length of 1984 bits. We chose the upper limit on the length of the bases to be equal to the length of the modulus[11]. To get a better overview, we split the computation time in a pre-computation and a policy-dependent part. The pre-computation consists of the computation of A' and the computation of $R_0^{\tilde{m}_0} S_1^{\tilde{v}_1} S_2^{\tilde{v}_2}$ as specified in Section 3.4. The timings presented are not only computations, but include communication times, i.e., they represent a real interaction with the card as it proves holdership of a credential. Communication time occurs while sending a number to the card or receiving the result from the card. We outline the result of these measurements in Table 2.

Table 2: Computation time comparison for different bit lengths of the modulus.

Modulus length	1280 bit	1536 bit	1984 bit
Pre-computation	5203ms	7828ms	13250ms
compute A'	2125ms	2906ms	5000ms
compute T_1	3078ms	4922ms	8250ms
Policy dependent	2234ms	2625ms	3298ms
compute s.	562ms	656ms	828ms
Total	7437ms	10453ms	16548ms

To get an idea which operations need a significant amount of the overall computation time we analyzed the time each individual operation consumes. The result of this experiment is listed in Table 3. We ran 1000 executions of the protocol's arithmetic operations to acquire the results. The upper bound on the bit length of the base was 1536 and the upper bound on the bit length of the exponent 895. We used a random 1536-bit number as modulus. For simplicity reasons, we rounded the percentages of computation times.

Table 3 shows that the addition, as a part of the multiplication and in various locations in the protocol, accounts for 31% of the overall computation time. The pseudo randomness generation accounts for roughly 9% of the computation time.

5. CONCLUSION

We present the first efficient implementation of an anonymous credential system on a standard Java Card. Our sys-

[9]Factories, such as our CredentialFactory, are the focus and control point for class instantiation and access. For instance, Credential instances cannot be constructed directly, but need to be created by the corresponding factory in a well-defined robust process.

[10]The Java Card 2.2.1 standard [26] can make transactions of multiple computations and write operations to persistent memory atomic. Either the entire transaction finishes successfully or the card is reset to the state before the transaction.

[11]In a setting with a large exponent (> 200 bits), the length of the base has a negligible impact on the computation time.

Table 3: Computation time comparison split up into the different low-level operations.

Operation	Time	# Ops	% (time)
Multiplication	4653ms	9	40%
Addition	2988ms	36	26%
ModSquare	243ms	27	2%
ModExp	4308ms	10	37%
Pseudo RNG	1088ms	16	9%
True RNG	815ms	1	7%
Addition	581ms	7	5%
Digest	220ms	10	2%
Total	11665ms		100%

tem nurtures sustainable secondary use of the user's identity because of the multi-use unlinkability of the credential system. Our system performs the entire computation on card and independently from a potentially malicious terminal. Therefore, we fulfill our major requirement for an autonomous trust root. In addition, our anonymous credential system offers long-term certificates and, therefore, does not require updates when doing many unlinkable proofs.

Our Java Card implementation is capable of efficient proofs of possession of identity credentials. Even though our implementation is able to include several attributes in a credential, we opt to trust the hardware for attribute statements, because this results in constant and low transaction times. We present this method as means to handle range proofs on a Java Card, as traditional Boudot range proofs are beyond reach of current cards. We propose to combine this with an efficient anonymous card revocation mechanism.

As limitations of our solution, we note that a terminal can attempt to send multiple policy requests to infer the user's data (see the autonomous trust root discussion in Section 2.1). We believe that this is orthogonal to the implementation of an anonymous credential system and requires further research. Even though our anonymous credential system provides multi-use unlinkability, we note the potential risk that a terminal may identify the Java Card by other means. A card could, for instance, contain further applets that disclose traceable information, such as a serial number. The card's hardware may also be traced by low-level information and finger printing.

In conclusion, we are confident that our solution has overcome the final technical barrier to establish privacy-preserving eID cards.

6. OUTLOOK

We present a possible extension of our approach that allows for efficiently proving multiple finite-set attributes. In addition, we throw a glance at the future of smart cards in general, and Java Cards in particular.

6.1 Camenisch-Groß Attribute Encoding

Camenisch and Groß [8] proposed a first approach to achieve higher efficiency with eID cards and CL signatures by encoding binary and finite set attributes in a single attribute base. This reduces the number of attribute bases and, therefore, of exponentiations by the number of prime-encodable attributes. In addition it allows for efficient *not*, *conjunction* and *disjunction* proofs. To be precise, they encode binary and finite set attribute values as prime numbers e_j, and condense them in a dedicated attribute base as product exponent $E = \prod_j e_j$, here at base R_1. To disclose a conjunction of prime-encoded values, one discloses the prime representation and proves knowledge of the remainder.

This method is currently realized for the Identity Mixer library on the PC and also suitable for the anonymous credential system on Java Card. In this case, the system requires one additional base for the prime-encoded attributes. AND-proofs of any number of binary or finite-set attributes will then cost one additional exponentiation.[12]

6.2 Future Smart Cards

Our system has very small hardware requirements. This allows us to implement the anonymous credential system on a smart card that is similar to those currently used in eID production. Our hardware is far from the upper end of the current technology spectrum. This supports our take-home message that todays eID cards are powerful enough to perform advanced privacy-preserving computations.

In addition, it gives us room to plan for the future. Whereas our current implementation copes with 16KB of EEPROM, 2KB of RAM, and a 3.57MHz 8-bit CPU[13], most recent cards are equipped with up to 1MB of EEPROM, 32KB of RAM and a 66MHz, 16-bit CPU[14]. Furthermore, the trends in smart card technology let us predict what smart cards may be chosen for future eID proposals. Those smart cards are clearly able to host further credential system features, such as verifiable encryption and e-cash-based frequency boundaries.

6.3 Java Card 3.0 Standard

The Java Card 3.0 standard as released in April 2008 by Sun Microsystems may also benefit our endeavor, because it allows Java Cards to hold RSA private keys in transient memory. Our current implementation makes use of a JCOP-specific library that allows us to store RSA keys in RAM instead of protected EEPROM, which has major influence on the computation times. With cards implementing the new standard, consequently, we can implement our construction fully standard-compliant on Java Card 3.0 cards. Such cards are to be released in 2009.

7. ACKNOWLEDGMENT

We would like to thank the BlueZ group from the IBM Research Zurich Lab for their continuous support and extremely helpful insight. In particular, we valued the discussions with Michael Baentsch (eID scenarios), Thomas Eirich (fast addition), Thorsten Kramp (fast computations on JCOP), Michael Kuyper (JCOP environment), Michael Osborne (eID scenarios, card support and personalization), Tamas Visegrady (fast computations and crypto acceleration on JCOP), and Thomas Weigold (JCOP environment, fast addition). Without their expertise in the domain of

[12]Based on the performance measurements of Table 3, we predict that the inclusion of Camenisch-Groß encoding and its AND-proofs will add 1684ms transaction time at a modulus bit length of 1536 bits. This accounts for the modular exponentiation, required multiplications, additions and PRNG calls. Of this, 1016ms are pre-computation, 668ms are policy-dependent. Note that these are upper bounds for the full exponent length.

[13]NXP JCOP v2.2/41

[14]Infineon, SLE88 family

smart cards and authentication solution, this work would have not been possible. We also appreciated the support of Doug Dykeman and Peter Buhler for this exploratory research project.

This work has been funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216483. Also, Victor Shoup has done his work at IBM Research and is supported by NSF award number CNS-0716690.

8. REFERENCES

[1] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In *Security and Cryptography for Networks*, vol. 4116 of *LNCS*, pages 111–125, Berlin, 2006. Springer.

[2] J. M. Balasch Masoliver. Smart card implementation of anonymous credentials. Master's thesis, K.U.Leuven, Belgium, 2008.

[3] P. Bichsel. Theft and misuse protection for anonymous credentials. Master's thesis, ETH Zürich, Switzerland, November 2007.

[4] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. K. Franklin, editor, *CRYPTO '04*, vol. 3152 of *LNCS*, pages 41–55. Springer, 2004.

[5] F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *EUROCRYPT '00*, vol. 1807 of *LNCS*, pages 431–444. Springer, 2000.

[6] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000.

[7] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proc. 11th ACM CCS*, pages 225–234. ACM Press, 2004.

[8] J. Camenisch and T. Groß. Efficient attributes for anonymous credentials. In *Proc. 15th ACM CCS*, pages 345–356. ACM Press, Nov. 2008.

[9] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT '01*, vol. 2045 of *LNCS*, pages 93–118. Springer, 2001.

[10] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. K. Franklin, editor, *CRYPTO '04*, vol. 3152 of *LNCS*, pages 56–72. Springer, 2004.

[11] J. Camenisch and E. Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Proc. 9th ACM CCS*. ACM Press, 2002.

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

[13] D. Chaum and J.-H. Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In M. Odlyzko, editor, *CRYPTO '86*, vol. 263 of *LNCS*, pages 118–167. Springer, 1987.

[14] Common Criteria Portal. Common criteria for information technology security evaluation. [online; 18 April 2009]. `http://www.commoncriteriaportal.org/`.

[15] I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. `http://eprint.iacr.org/2001`, 2001.

[16] L. Danes. Smart card integration in the pseudonym system Idemix. Master's thesis, University of Groningen, 2007.

[17] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO '86*, vol. 263 of *LNCS*, pages 186–194. Springer, 1987.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995. Elements of reusable object-oriented software.

[19] X. Huysmans. Privacy-friendly identity management in eGovernment. In *The Future of Identity in the Information Society*, vol. 262/2008 of *IFIP International Federation for Information Processing*, pages 245–258. IFIP, Springer, June 2008.

[20] IBM. JCOP - the IBM GlobalPlatform JavaCard$^{\text{TM}}$implementation. [online; 16 April 2009], Feb. 2002. `ftp://ftp.software.ibm.com/software/pervasive/info/JCOP_Family.pdf`.

[21] IBM. Cryptographic protocols of the Identity Mixer library, v. 1.0. IBM Research Report RZ3730, IBM Research, 2009. `http://domino.research.ibm.com/library/cyberdig.nsf/index.html`.

[22] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. Heys and C. Adams, editors, *Selected Areas in Cryptography*, vol. 1758 of *LNCS*. Springer, 1999.

[23] Philips. mifare proX P8RF5016. [online; 18 April 2009], May 2003. `http://smartdata.usbid.com/datasheets/usbid/2005/2005-q2/sfs051814.pdf`.

[24] M. O. Rabin and J. O. Shallit. Randomized algorithms in number theory. *Communications in Pure and Applied Mathematics*, 39:239–256, 1986.

[25] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM*, 21(2):120–126, Feb. 1978.

[26] Sun Microsystems. Java card platform specification 2.2.1. [online; 18 April 2009], Oct. 2003. `http://java.sun.com/javacard/specs.html`.