Noname manuscript No.
(will be inserted by the editor)

Beyond the selective disclosure of ABCs on RAM-constrained devices

Antonio de la Piedra

Received: date / Accepted: date

Abstract The utilization of private Attribute-based credentials (ABC) in everyday life could enable citizens to only partially reveal their identity in economic transactions and communication with public institutions. This means citizens could control in a practical way the information related to their own life and identity in many contexts. At the time of writing, the Identity Mixer (Idemix) by IBM is the only credential system that offers enough flexibility to proof a considerable variety of properties of the attributes of a credential. Despite many practitioners have proposed different strategies for implementing ABCs on smart cards in the last few years, the complexity of the assumptions these primitives usually rely on, undermines fast and practical implementations of ABCs. The lack of smart cards with powerful hardware arithmetic accelerators is not the only problem for speeding up the computation of these primitives since one need to perform fast arithmetic operations with operands stored in RAM. Moreover, the implementation of complex Zero-Knowledge Proofs (ZKP) needs a considerable amount of pseudorandomness. In order to overcome these limitations, we proposed to use a Pseudo-Random Number Generator (PRNG) for recomputing pseudorandomness and we use it tandem with variable reconstruction in order to implement complex proofs. The utilization of this simple technique enable us to compute pseudonyms, domain pseudonyms, multi-credential proofs and to rely on the AND, NOT and OR operators to prove inner properties of the attributes of the credential whereas prior art only addressed the selective disclosure of one attribute on a given credential. Moreover, we show how to increase the number of attributes stored on the card via this construction. Finally, we show how to chain proofs based on AND, NOT and OR operators in order to extend the amount of properties of a credential that can be showed via external and internal commitment reordering.

Keywords Attribute-based credentials · Smart Cards · Privacy

Radboud University Nijmegen, ICIS DS, Nijmegen, The Netherlands.
E-mail: a.delapiedra@cs.ru.nl

1 Introduction

Generally, purchasing adult goods such as alcohol, cigarettes and require an identification procedure between the buyer and the seller. We provide our identity in order to prove this when only our age would be enough in the majority of the cases. Besides, certain transactions such as opening a back account require our full identity whereas a registration number could be enough. Further, these two real life examples can be easily translated to on-line transactions.

There is no need to reveal our full identity if a citizen can rely on primitives such as ABCs. The selective disclosure of attributes is typically the main operation of ABCs where only a set of attributes of a certain identity needs to be revealed in order to fulfill an authorization operation whereas the representation of the others are proved in zero knowledge. This operation is generally related to a presentation policy i.e. a list of attributes that a prover must fulfill [8]. This is the main operation in many schemes proposed in the literature such as U-Prove [5], ACL [2] and Idemix [11].

However, it can be useful to proof that more than one credential contains the same attribute, to assert a certain property of an attribute in relation to others

such as my credential does not contain a certain attribute or to create multiple pseudonyms for a given identity so a user can temporary link their identity to a certain domain. In order to implement this complex procedures we rely on different strategies that we present in this manuscript. In so doing, we rely on the only open-source and practical implementation of Idemix, the I Reveal My Attributes (IRMA) card[1].

In the next section, we describe the work of other practitioners related to the implementation of ABCs on smart cards and put their performance figures in relation with our contributions in this manuscript. In Section 3 we describe the main building blocks of ABCs: non-interactive commitment schemes, the Camenisch-Lysyanskaya (CL) digital signature and the structure of a private ABC system. In Section 4, we sketch out the internals of the IRMA card and present two execution models of the verification and the issuing process of Idemix that we use to optimize the current implementation. Section 5 describes the impact of PRNG in an ABC compact implementation for recomputing pseudorandomness in proofs of knowledge. This is the main component together with variable reconstruction in RAM and commitment reordering that we rely on for making it possible the execution of complex and multi-credential proofs on smart cards. Section 6 and 7 include the strategies we utilized for executing complex proofs based on traditional and prime-encoded attributes. In Section 8 we present the open-source framework we developed for obtaining our performance figures and we end in Section 8 with some conclusions.

2 Related work

ABCs are cryptographic constructions that work over a credential, a container of attributes, that is signed by a blind and randomizable signature [17]. By selectively disclosing these attributes, some of them, ones proves properties about its identity. The main arithmetic operation behind this schemes is the modular exponentiation[2].

The availability of smart cards in the market that could support the type of arithmetic that ABCs constructions relies on, is translated in first implementations that only supported one credential with one attribute. Thus, most of the proposed implementations rely on Java Card and MULTOS. In the last few years, several credential systems have been implemented on smart cards such as Idemix [4, 28, 29], U-Prove [27] and ACL [23].

For those whose main operation is a modular exponentiation the main challenge in literature was to perform this operation which the restrictions applied to the size of parameters in platforms where modular arithmetic is not flexible. This is the case of both Java Card and MULTOS, where the RSA encryption[3] operation is abused [4, 28, 29] in order to: (1) ease the computation of partial exponentiations (2) make it possible the exponentiation with large exponents which cannot be used by the Java Card and MULTOs API.

However, this strategy is still too slow on Java Cards as the performance of figures Bichsel et al. (1,280-bit modulus) and Sterckx et al. (1,024-bit modulus) reveal [4, 28]. For proving the ownership of a signature over a credential of only one attribute they required 7.4 and 4.2 seconds respectively.

Practical results for the issuing and selective disclosure of attributes on smart cards using Idemix were not possible till 2013, where Vullers et al. obtained performances figures of 1–1.5 seconds of both operations relying on credentials on up to 5 attributes. This implementation resulted in the IRMA card [29]. In this manuscript we rely on that implementation in order to propose further strategies for computing complex proofs on smart cards or embedded devices with a small amount of RAM.

2.1 Our contribution

As we showed in the prior section, the implementation of credential systems on smart cards in the literature is usually restricted to proving the ownership of a small amount of attributes due to the computational complexity of these protocols [4,28,29]. Moreover, the available amount of RAM is generally limited in smart cards and relying on it for performing arithmetic operations is crucial for speeding up the verification process. In this manuscript we present[4]:

- The design of a PRNG for reducing the RAM requirements of the execution of proofs of knowledge on smart cards (Section 5).

[1] `https://www.irmacard.org`

[2] This is the case of Idemix, whose main component, the Camenisch-Lysyanskaya signature relies on the Strong RSA assumption.

[3] As described by [28], an exponent e whose length goes beyond the limits of the smart card API can be split in a set of l sub-exponents $e = e_{l-1}||...||e_0$ of length d so it can be reconstructed as $\sum_{i=0}^{l-1} e_i \cdot 2^{l \cdot d}$. Hence, $a^e \mod n$ is performed as $\prod_{i=0}^{l-1} a^{e_i \cdot 2^i \cdot d} \mod n$ for l sub-exponents of length d and base a.

[4] Our performance figures have been extracted from a MULTOS ML3-R3-80K smart card using the SCM Microsystems SCL011 reader in a Intel Core i5-3230M CPU clocked at 2.60GHz running Debian Linux 3.13.6-1, python 2.7.6, python-pyscard 1.6.12.1-4 and CHARM 0.43 [1].

– How to perform proofs involving pseudonyms, domain pseudonyms and multi-credential proofs (Section 6).
– Idemix can rely on attributes encoded as prime numbers. In so doing, proving properties of a credential relying on the AND, NOT and OR operators is possible [10]. We describe our implementation options and present how to combine them through the reorganization of the commitments involved (Section 7).

3 Preliminaries

In this section we describe the building blocks of Idemix (non-interactive commitment schemes, blind signatures and zero-knowledge proofs) as we rely on them for describing our optimizations across this manuscript.

Non-interactive commitment schemes. Idemix relies on Fujisaki-Okamoto commitments [22], which are statistically hiding and computationally binding when factoring is a hard problem. A user proves the knowledge of a committed value such as an attribute that is not revealed during the issuing and verification operation. Given an RSA special modulo n, $h \in QR_n$ and $g \in < h >$, the commitment function for an input x, and random value $r \in Z_n$ is computed as $g^x h^r \mod n$.

Zero-knowledge proofs. In a proof of knowledge, a verifier is convinced that a witness w satisfies a polynomial time relation R only known by the prover. If this is performed in a way that the verifier does not learn w, this is called a zero-knowledge proof of knowledge. Damgård proved that is possible to generate zero-knowledge protocols via sigma protocols [19]. In Idemix, the typical three movement of sigma protocols (commitment, challenge and response[5]) is transformed into a Non Interactive Proof of Knowledge (NIZK) via the Fiat-Shamir heuristic [21] in the random oracle model. A variety of zero-knowledge protocols are utilized in Idemix. For instance, proofs of knowledge of discrete logarithm representation modulo a composite are used during issuing and verification [22].

A variety of zero-knowledge protocols are utilized in Idemix. For instance, proofs of knowledge of discrete logarithm representation modulo a composite are used during issuing and verification [22]. Besides, it is also possible to rely on proofs of knowledge of equality of representation for proving, for instance, that a certain amount of attributes in two credentials share the same value [14].

The CL digital signature scheme. This primitive is main block of Idemix [13]. It provides multi-show unlinkability via the randomization of the issued signature. That means that the cardholder cannot be traced when proving the ownership a signature over a certain amount of attributes. A digital signature scheme is composed of three PPT algorithms: a key generation algorithm Gen, a signature creation primitive Sign and a verification technique Verify over a message m. In this scheme, $\forall m \in M$, given a certain public key (PK) and a secret key (SK), Sign(SK, m) is always accepted by the Verify algorithm.

This signature is secure under the *Strong RSA* assumption. A CL signature is generated (Gen) by a certain issuer according to her public key $(S, Z, R_0, R_1, ..., R_5 \in QR_n, n)$ using its secret key (p, q). For instance, a CL signature over a set of attributes $(m_0, ..., m_5)$ is computed by selecting A, e and v s.t. $A^e = ZR_0^{-m_0} R_1^{-m_1} R_2^{-m_2} R_3^{-m_3} R_4^{-m_4} R_5^{-m_5} S^{-v} \mod n$. Then, a third party can check the validity of the signature by using the issuer's public key and the triple (A, e, v) as $Z \equiv A^e R_0^{m_0} R_1^{m_1} R_2^{m_2} R_3^{m_3} R_4^{m_4} R_5^{m_5} S^v \mod n$ (Verify).

Private ABC systems. They were firstly proposed by Chaum in [18]. In these systems, users remain anonymous and are only known by their pseudonyms. There are organization that issue and verify credentials so a user can prove its identity and avoid colluding between them. In Idemix, this is performed via the multi-show unlinkability property of the CL digital signature scheme. Typically, is crucial to avoid the transference of credentials between users. This is enforced by using a secret key that is only known to the user and not by the system (namely, a master secret m_0 in Idemix). In this system, there are two main protocols: issuing (or GrantCred [12]) and verification (or VerifyCred [12]). In the first one, a certain cardholder performs a protocol for signing a committed value, for instance, a set of attributes that represent her identity e.g. $m_0, ..., m_l$ for l attributes. At the end of the protocol, she receives a signature σ whereas the signer did not learn anything about $m_0, ..., m_l$.

On the other hand, the verification operation serves for proving the knowledge of a signature over a committed value, for instance a set of attributes and the master secret m_0 of the user for a pair cardholder/verifier. This protocol enables the possibility of using policies (see for instance [7]), i.e. a list of attributes or conditions in a certain credential that must be fulfilled during an authentication operation. Accordingly, an empty proof of possession over a set of attributes $(m_0, ..., m_5)$ is represented using the Camenisch-Staedler notation [16] as:

[5] In the first stage, the prover sends to the verifier a commitment message t or t_value. In the second move, the verifier sends to the prover a random challenge message c. Finally, the last message sent by the prover includes a response value or s_value.

NIZK: $\{(\varepsilon', \nu', \alpha_0, ..., \alpha_5) : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \mod n\}$ being the Greek letters (ε', ν') and $(\alpha_0, ..., \alpha_5)$ the values of the signature and the set of attributes proved in zero knowledge and not revealed i.e. $\in A_{\bar{r}}$. The set of revealed attributes is represented by A_r. Similarly, one can prove the CL signature over a set of attributes revealing some of them. For instance, revealing m_1 and hiding $(m_0, m_2, m_3, m_4, m_5)$ would be represented in zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_2, \alpha_3, \alpha_4, \alpha_5) : Z R_1^{-m_1} \equiv \pm R_0^{\alpha_0} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \mod n\}$.

In Idemix, each credential can be categorized as an attribute container, protected by a CL signature generated by an issuer. This signature guarantees the integrity of the credentials i.e. modification, deletion or adding new attributes to a credential by the user can be easily detected by a verifier. Moreover, each credential is linked to the cardholder by her master secret, securely stored on the card in IRMA. Finally, other protocols for generating a pseudonym associated to a credential are desirable in this type of systems (Section 6.1), as well as proving properties of attributes within a credential (Section 7) or across multiple credentials (Section 6.2).

4 The IRMA card

In this section we describe how the issuing and verifying operation of Idemix are implemented on the IRMA card. For both operations a latency model is created so we can related to it in the coming sections, where our optimizations are explained.

IRMA is based on the MULTOS card. Particularly, the target device is the ML3-80K-R1 version. It is based on the SLE 78CX1280P chip by Infineon[6]. This processor, clocked up to 33 MHz, provides an instruction set compatible with the Intel 8051 and hardware accelerators for ECC, RSA, 3DES and AES.

4.1 Issuing in IRMA

Issuing in Idemix is related to the generation of a CL blind signature over the attributes of the cardholder. In so doing, the issuer cannot extract the master secret m_0 of the cardholder and the generated tuple (A, e, v) remains hiden too. However, in IRMA the cardholder's attributes are never revealed to the issuer.

[6] `http://www.infineon.com/dgdl/SP0\_SLE+78CX1280P\_2012-07.pdf?folderId=db3a304325afd6e00126508d47f72f66&fileId=db3a30433fcce646013fe3d672214ab8` (Accessed 6 January 2016)

IRMA supports up to 5 attributes by credential and relies on 1,204-bit special RSA modulus for performance reasons[7].

Issuing requires two NIZKs (Table 1). In the first one, the cardholder proves to the user that she knows the representation of U, v' and her master secret m_0 as NIZK: $\{(\nu', \alpha_0) : U \equiv \pm S^{\nu'} R^{\alpha_0} \mod n\}$. Afterwards, the issuer proves the knowledge of $1/e$ as NIZK: $\{(1/e) : A \equiv \pm Q^{e^{-1}} \mod n\}$. Finally, the cardholder verifies that the signature (A, e, v) blindly generated by the issuer is correct as $Z \equiv A^e R_0^{m_0} R_1^{m_1} R_2^{m_2} R_3^{m_3} R_4^{m_4} R_5^{m_5} S^v \mod n$.

Table 1 Message flow for issuing a CL signature over a set of attributes (I: Issuer, C: Cardholder)

Common inputs: The public key of the issuer $(S, Z, R_0, R_1, ..., R_5 \in QR_n, n)$, the number of attributes that are issued.
Cardholder inputs: The credential involved and its set of attributes $m_0, ..., m_5$.
Issuer inputs: The private key of the issuer p, q.

Protocol: The cardholder first proves the knowledge of the representation of U as NIZK: $\{(\nu', \alpha_0) : U \equiv \pm S^{\nu'} R^{\alpha_0} \mod n\}$. Then, the issuer proves the knowledge of $1/e$.

1. $I \rightarrow C$: The user chooses a random nonce $n_1 \in_R \{0,1\}^{l_\diamond}$ and sends it to the cardholder.
2. $C \rightarrow I$: The cardholder computes the commitment $U = S^{v'} R^{m_0} \mod n$ with $v' \in \pm\{0,1\}^{l_n + l_\diamond}$. It proves in zero knowledge m_0 with a randomizer α_0, v'. Then ,it generates the s_values for $\hat{v}'$, $\hat{r}$ together with the challenge c with the context, U and the nonce n_1. Finally, it sends to the issuer $n_2 \in_R \{0,1\}^{l_\diamond}$.
3. $I \rightarrow C$: It verifies the NIZK as $\hat{U} = U^{-c}(S^{\hat{v}'} R_0^{\hat{\alpha_0}}) \mod n$. Then, it proves the knowledge of $\frac{1}{e}$ via random values e, $\tilde{v}$, v''. It computes the commitments $Q = Z\ U^{-1} S^{-v''} \mod n, A = Q^{e^{-1} \mod p'q'} \mod n$. It sends (A, e, v'') and creates the proof NIZK: $\{(1/e) : A \equiv \pm Q^{e-1} \mod n\}$. It computes $\tilde{A} = Q^r \mod n$ for $r \in_R Z_{p'q'}$, and obtains the challenge c' as the hash of the context, Q, A, n_2 and $\tilde{A}$. It computes $S_e = r - c' \cdot e^{-1} \mod p'q'$ and sends A, e, v'', S_e, c' to the cardholder.
4. C : Computes $v = v'' + v'$ and verifies (A, e, v) as $Q = Z\ S^{-v} R_0^{m_0} \mod n$ with $\hat{Q} = A^e \mod n$.

[7] The attributes are represented as $l_m = 256$ bits. The rest of parameters are set as $l'_e = 120$, $l_\diamond = 80$, $l_H = 256$, $l_e = 504$, $l_n = 1,024$, $l_e = 597$, and $l_v = 1,604$ bits. The term l'_e represents the size of the interval where the e values are selected, $l_\diamond$ is the security parameter of the statistical ZKP, l_n represents the size of the RSA modulus, l_e is the size of e values of the certificates, and l_H is the domain of the hash function used in the Fiat-Shamir heuristic (we use SHA-256). Finally, l_e and l_v are related to the size of e and v parameters of the CL signature.

In IRMA, the issuing part of Idemix mimics the Cardholder-Issuer interaction as a set of states: `ISSUE_CREDENTIAL`, `ISSUE_PUBLIC_KEY`, `ISSUE_ATTRIBUTES`, `ISSUE_COMMITMENT`, `ISSUE_COMMITMENT_PROOF`, `ISSUE_CHALLENGE`, `ISSUE_SIGNATURE` and `ISSUE_VERIFY`. The first, state `ISSUE_CREDENTIAL`, puts the card in in issuance mode, sends the identifier of the credential that will be issued and the context of the operation. Then, during the `ISSUE_PUBLIC_KEY` state, the card accepts the public key of the issuer: $n, S, Z, R_0, ..., R_5$. The attributes to be issued are sent to the card in the `ISSUE_ATTRIBUTES` state. The rest of the states are related to the execution of the two NIZKs.

During `ISSUE_COMMITMENT` the cardholder receives the nonce n_1, it computes U and returns it. Then, in `ISSUE_COMMITMENT_PROOF`, the required values for proving the knowledge of m_s in U: $c, \hat{v}'$, $\hat{s}$ are generated. In `ISSUE_CHALLENGE`, it sends n_2. During the `ISSUE_SIGNATURE` mode, the issuer constructs the blinded CL signature and sends to the card the partial signature (A, e, v''). Finally, in `ISSUE_VERIFY` the card verifies the signature using the values sent the verifier (c, S_e).

4.1.1 Execution model

We can model the latency of the issuing process in the IRMA card by representing the time required for performing the operation described in Table 1 as $T_{issuing}(n)$ where n is the number of attributes that will be issued in a certain credential. This latency would be result of summing up the time required for getting the public key of the issuer, adding the computation of the involved proofs and the process of obtaining and verifying the signature:

$$T_{issuing}(n) = T_{sel\_cred} + \sum_{i=n,S,Z,R_i} T_{get\_PK}(i) + \sum_{i=1}^{n} T_{get\_attr}(i) + T_{gen\_commitment} + \sum_{i=c,\hat{v}',\hat{s}} T_{gen\_proof}(i) + \sum_{i=A,e,v''} T_{get\_signature}(i) + T_{verify}(n) \quad (1)$$

From this model, we know that there are only two operations that depends on the number of attributes issued that are part of a certain credential: $T_{get\_attr}(i)$ and $T_{verify}(n)$. That would mean that in order to optimize the overall latency of $T_{issuing}(n)$ there are two strategies: (1) reduce the number of attributes that are part of the credential (we revisit this aspect in Section 7) and (2) reduce then number of operations in the verification part of the proof, which is already implemented on the IRMA card where the second proof is optionally verified for reducing the computational complexity of the operation.

4.2 Verification in IRMA

As in the issuing operation, the Idemix prover is modelled in the IRMA card as set of states (`PROVE_CREDENTIAL`, `PROVE_COMMITMENT`, `PROVE_SIGNATURE` and `PROVE_ATTRIBUTE`) that makes it possible the Prover-Verifier interaction described in Table 2.

A presentation policy with a list of conditions the smart card must fulfill is sent during the `PROVE_CREDENTIAL` state. This consists on a list of attributes that will be revealed ($\{m_i\}_{i\in A_r}$) and hidden($\{m_i\}_{i\in A_{\bar{r}}}$). Subsequently, during the `PROVE_COMMITMENT` state, the card performs the operations described in Table 2. Finally, the verifier can request the card both the randomized tuple $(A', \hat{e}, \hat{v}')$ and the attributes that are either revealed or hidden (states `PROVE_SIGNATURE` and `PROVE_ATTRIBUTE` respectively).

4.2.1 Execution model

We designed a latency model of the Idemix verification based on the number of attributes per credential (n) that are revelaled (r) or hidden so in the worst case $n - r + 1$ computations[8] are required for generating each $\hat{m}_i$.

Eq. 2 represents the overall latency of the four states described above according to the (n, r) parameters.

$$T_{verify}(n, r) = T_{sel\_cred} + T_{gen\_commit}(n, r) + \sum_{i=A,e,v} T_{get\_sig}(i) + \sum_{i=1}^{n} T_{get\_attr}(i) \quad (2)$$

The time the state `PROVE_CREDENTIAL` takes is related to T_{sel_cred}, `PROVE_COMMITMENT` is represented by $T_{gen\_commit}(n, r)$. Finally, the time related to the `PROVE_SIGNATURE` and `PROVE_ATTRIBUTE` states (where the verifier can ask for the randomized triple and the revealed and hidden attributes) has to do with $T_{get\_sig}(i)$ and $T_{get\_attr}(i)$ respectively.

Furthermore, the latency of the `PROVE_COMMITMENT` state can be expanded to the following expression:

$$T_{gen\_commit}(n, r) = \sum_{i=A,v} T_{rand\_sig}(i) + T_{gen\_t\_values}(n - r + 1) + T_{hash} + T_{gen\_s\_values}(n - r + 1) \quad (3)$$

[8] One extra computation is always considered since the master secret is always hidden.

Table 2 Message flow for proving the ownership of a CL signature over a set of attributes (V: Verifier, C: Cardholder)

Common inputs: The public key of the issuer $(S, Z, R_0, R_1, ..., R_5 \in QR_n, n)$, the presentation policy i.e. those attributes that must be revealed $m_i \in A_r$ w.r.t. to those that can be hidden $m_i \in A_{\bar{r}}$.
Cardholder inputs: The credential involved and its set of attributes $m_0, ..., m_5$, randomizers and the tuple (A, e, v) over $m_0, ..., m_5$.
Verifier inputs: $A', v', m_i \in A_r, m_i \in A_{\bar{r}}$

Protocol: The (cardholder, verifier) pair perform the following NIZK where the cardholder proves the ownership of a CL signature over $m_0, ..., m_5$ and reveals, for instance m_1: NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_2, \alpha_3, \alpha_4, \alpha_5) : ZR_1^{-m_1} \equiv \pm R_0^{\alpha_0} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \mod n\}$.

1. $V \rightarrow C$: The verifier sends a fresh nonce n_1 to the verifier together with a presentation policy that must be fulfilled.
2. $C \rightarrow V$: Using the tuple (A, e, v) over $m_0, ..., m_5$, it randomizes the signature by generating $r_A \in_R \{0,1\}^{l_n + l_\varnothing}$ and obtaining the new tuple (A', e', v') as $A' = AS^{r_A} \mod n$, $v' = v - er_A$ and $e' = e - 2^{l_e - 1}$. Afterwards, it generates the randomizers $\tilde{e} \in_R \pm\{0,1\}^{l'_e + l_\varnothing + l_H}$, $\tilde{v}' \in_R \pm\{0,1\}^{l_v + l_\varnothing + l_H}$, and $\tilde{m_i} \in_R \pm\{0,1\}^{l_m + l_\varnothing + l_H} (i \in A_{\bar{r}})$ and computes the commitment (t-value) $\tilde{Z} = A'^{\tilde{e}} (\prod_{i \in A_{\bar{r}}} R_i^{\tilde{m_i}}) S^{\tilde{v}'}$. Then, it generates the challenge c by hashing the context, the t_value and the nonce. It generates the response values (s-values) $\hat{e} = \tilde{e} + ce'$, $\hat{v}' = \tilde{v}' + cv'$ and $\hat{m_i} = \tilde{m_i} + cm_i (i \in A_{\bar{r}})$. Finally, It sends the challenge c, the common value A', the response values and $m_i \in A_r$ to the verifier.
3. V: Verifies if the proof is correct via the issuer public key by computing $\hat{Z} = (Z\, A'^{-2^{l_e - 1}} \Pi_{i \in A_r} R_i^{-m_i})^{-c} (A')^{\hat{e}} (\Pi_{i \in A_{\bar{r}}} R_i^{\hat{m_i}}) S^{\hat{v}'})$ and checking if c equals the hash of A', $\hat{Z}$ and n_1.

The randomization of the CL signature and the generation the respective *t_values*, *s_values* and challenge c of the proofs is represented as:

$$T_{gen\_s\_values}(n - r + 1) = T_{gen\_\hat{e}} + T_{gen\_\hat{v}'} + \sum_{i=0}^{r} T_{gen\_\hat{m}_i} \quad (4)$$

Hence, $T_{gen\_t\_values}$ is related to the computation of the commitments according to the number of non-disclosed attributes: $\Sigma_{i=1}^{n-r+1} T_{mul\_exp}(R_i^{\tilde{m_i}})$.

In the next section we will rely on the model in order to extend the number of attributes of the IRMA card. In Section 7.1.2 we study how to optimize the issuing operation via the model described in Section 4.1.

5 Performance impact of a PRNG for regenerating pseudorandomness

Our target device comprises 960 bytes of RAM and additional 1,160 bytes of transient memory that can be abused under certain conditions e.g. when the APDU does not overlap with the amount of data stored in that part of the memory. Increasing the number of attributes as well as performing complex proofs on the smart card has do to with generating and storing a considerable amount of pseudornadomnnes. In the case of increasing the number of attributes, this has to do with generating and storing $\hat{m_i}$ values.

In the current implementation of IRMA, based on $5 + 1$ $\hat{m_i}$ attributes (taking into account the master secret, which is always hidden), one verification session requires $74 \cdot 6 = 444$ bytes of RAM, 74 bytes is the required space for storing one $\hat{m_i}$ value. Therefore, any optimization should be based on rearranging the storage of the pseudorandom data required in the proofs.

5.1 Rationale

If we take a look to Equations 2-4 in Section 4.2.1, the pseudorandomness used to derive the $(\tilde{e}, \tilde{v}')$ tuple and the $\tilde{m_i}$ values are used in both $T_{gen\_t\_values}$ and $T_{gen\_s\_values}$. Besides, time required for verifying a credential is dominated by the number of non-revealed attributes $(n-r)$ that requires: (1) the modular exponentiations computed during the generation of the *t_values* and (2), the pseudorandom generation of the $\tilde{m_i}$ values and the computation of the correspondent $\hat{m_i}$ value as $\hat{m_i} = \tilde{m_i} + cm_i$. All in all, any possible optimization in the implementation must be driven by: (1) recomputing the pseudorandomness utilized in the $\tilde{m}$ values and (2) reducing the overhead of the required computation for hiding the selected attributes. What we need is to generate as many $\hat{m_i}$ values as needed without being limited by the current amount of RAM.

Bichsel et al. suggested the utilization of a PRNG to regenerate the random exponent of the Idemix verification operation in the case of one credential with one attribute (i.e. the master secret) [4]. We rely on this idea as starting point and generate all the involved pseudorandomness, not only the random exponents for increasing the number of attributes on card and supporting pseudonyms, domain pseudonyms and prime-encoded proofs (AND, NOT and OR operators). Besides, with combine that approach with variable reconstruction in RAM so computing multi-credential proofs such as equality proofs of representation is also possible.

5.2 Design

We obtain performance figures of several PRNGs proposed in the literature [3,20] such as Fortuna, `HMAC_DBRG` and `HASH_DBRG` (Table 3). We restricted ourselves to these constructions because they rely on block ciphers and Message Authentication Codes (MACs), primitivites that are available in the our target device. Relying on these primitives, conjectured as pseudorandom generators under the assumption that one-way functions exist (cf. [24, 26]), we can expect that their output of the PRNG is indistinguishable from random by any PPT algorithm or distinguisher.

We have represented the performance figures of computing a $\hat{m}_i$ value of 74 bytes together with the respective number of calls to the GF in each case to generate $|\hat{m}_i|$ bytes. The performance of both using AES-128 in the Fortuna PRNG and SHA-1 is almost equal but when generating a considerable amount of pseudorandomness 3 ms can be significant[9].

Our aim is to provide backtracking resistance between verification sessions. In so doing, in each session a seed k is generated as the last 128 bits of the SHA-1 hash operation of the concatenation of the MULTOS PRNG output together with the string "IRMA". Then, we feed the seed into the GF (AES-128) as the key. Afterward a counter c is initialized to 0 and incremented in the generation of each pseudorandom block of 128 bits. When the verification process is finished the seed stored in RAM is erased by the discharge of the capacitors of the smart card and in the next verification session, a new seed is generated. Furthermore, prediction resistance is ensured if we rely on the security of the AES block cipher.

For each primitive where the PRNG is involved, we describe the sequence of pseudorandom values that are regenerated as: $init_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow reset_{PRNG}() \Rightarrow \tilde{m}_i$ where m_i in this case is the pseudorandomness associated to the selective disclosure of attributes.

5.3 Overhead

In the current implementation of the IRMA card, $74 \cdot 6$ bytes of RAM are reserved for storing all the $\hat{m}_i$ involved values. After the inclusion of the PRNG we only to main in the RAM: the seed/AES key ($128/8 = 16$ bytes), the counter c (16 bytes) and 74 bytes for the $\hat{m}_i$ values that are generated when required. Thus, when an $\hat{m}$ value needs to be generated, we get as many blocks as needed for filling the 74 random bytes space and the counter c is incremented after each block is computed. This requires $74 + 16 + 16 = 106$ bytes of RAM instead of $74 \cdot 6 = 444$ bytes. Nonetheless, an extra latency for computing all the $\hat{m}$ values at run time is expected (Table 4)[10].

[9] E.g. the number of required calls in our approach for performing equality proofs of representation (Table 4). If all the attributes are hidden the generation of the pseudorandomness associated to a $(A', \hat{e}, \hat{v}')$ triple for one credential of 5 attributes needs $9 + 16 + 4 + 5 \cdot 6 = 59$ calls to the PRNG.

6 Performance evaluation of complex proofs via traditional attributes

The PRNG we described in Section 5 that enable us to increase the number of attributes per credential can be also used for performing complex proofs on embedded devices with a small amount of RAM. In this respect, we show to compute standard pseudonyms, domain pseudonyms and equality proofs of representation using traditional attributes.

6.1 Standard and domain pseudonyms

In an ABC such as Idemix, pseudonyms guarantee a credential holder to be recognized the next time she visits a certain service provider (SP). The first type of pseudonym, that we call standard pseudonym consist of a randomized commitment to the master secret: $\mathsf{Nym} = g^{m_0} h^r \mod \Gamma$. Parameters (g, h) are two generators that together with the modulo Γ are part of the public system group parameters.

Besides, an organization can create pseudonyms associated to its domain. These pseudonyms are derived computed as $\mathsf{dNym} = g_{dom}^{m_0}$. In this case, $g_{dom} = \mathcal{H}(\mathsf{dom})^{(\Gamma-1)/\rho}$ and the group Z_Γ^* has order $\Gamma - 1 = \rho \cdot b$ for a prime ρ [12]. Proving the ownership of both a standard and domain pseudonyms can be performed in zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, ..., \alpha_5, \psi) : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \mod n \wedge \mathsf{Nym} \equiv g^{\alpha_0} h^{\psi} \mod \Gamma \wedge \mathsf{dNym} \equiv g_{dom}^{\alpha_0} \mod \Gamma\}$ without revealing any attribute.

6.2 Equality proofs of representation

A cardholder with several credentials can need to prove that all the credentials are of his own i.e. that every master secret is the same. This can be performed using equality proofs of representation [14]. For instance, if we have two credentials signed by two different issuers

[10] We use the notation RA for the cases where we reveal every attribute with the exception of the master secret and HA where every attribute is hidden.

Table 3 Performance of PRNG candidates in the IRMA card for generating an $\hat{m}_i$ value of 74 bytes

Work	PRNG	GF	Block size (bytes)	No. calls (GF)	Delay (ms)
[3]	`HMAC_DBRG`	SHA-1	20	12	48.64
[3]	`HASH_DBRG`	SHA-1	20	5	26.33
[3]	`HMAC_DBRG`	SHA-256	32	10	106.78
[3]	`HASH_DBRG`	SHA-256	32	4	47.47
[20]	Fortuna	AES-256	16	5	29.32
This work	Fortuna	AES-128	16	5	24.76

Table 4 Performance figures of complex proofs via traditional attributes

Implementation	Case	T_{sel_cred}	T_{gen_commit}	$T_{get\_sig}(A,e,v)$	T_{get_attr}	Total (ms)
Selective disclosure [29]	RA	104.12	826.25	45.14	69.55	1,045.10
Selective disclosure [29]	HA	105.20	1,259.24	47.07	72.78	1,484.32
Selective disclosure (this work)	RA	103.10	840.73	45.18	95.87	**1,084.91**
Selective disclosure (this work)	HA	104.95	1,307.35	46.08	229.80	1,688.20
Nym	RA	103.10	1,176.02	46.28	161.11	**1,486.51**
Nym	HA	103.15	1,647.33	46.46	293.51	2,065.42
Nym $\wedge$ dNym	RA	103.01	1,373.37	46.04	221.05	**1,743.54**
Nym $\wedge$ dNym	HA	104.23	1,836.00	46.02	351.70	2,338.01
Eq. proof (n cred.)	RA	104.12	1,805.24	702.10	133.07	2,744.51
Eq. proof (n cred.)	HA	105.23	2,738.78	695.14	344.65	3,883.83
Eq. proof (2 cred.)	RA	103.99	1,743.37	273.60	137.15	**2,261.19**
Eq. proof (2 cred.)	HA	103.47	2,673.60	266.80	347.11	3,390.60

(w.r.t. (A_1, e_1, v_1) and (A_2, e_2, v_2)) over our master secret m_0 we can describe an equality proof of representation without hiding every attribute as NIZK: $\{(\varepsilon'_1, \nu'_1, \varepsilon'_2, \nu'_2, \mu, (\alpha_1, ..., \alpha_5), (\beta_1, ..., \beta_5)) : Z^{(1)} \equiv \pm R_0^{(1)\mu} R_1^{(1)\alpha_1} R_2^{(1)\alpha_2} R_3^{(1)\alpha_3} R_4^{(1)\alpha_4} R_5^{(1)\alpha_5} A^{(1)\varepsilon'_1} S^{(1)\nu'_1} \mod n_1 \wedge Z^{(2)} \equiv \pm R_0^{(2)\mu} R_1^{(2)\beta_1} R_2^{(2)\beta_2} R_3^{(2)\beta_3} R_4^{(2)\beta_4} R_5^{(2)\beta_5} A^{(2)\varepsilon'_2} S^{(2)\nu'_2} \mod n_2\}$. In that proof, μ represents the non-disclosed master secret[11] and the two public keys of the issuers consists of $(S^{(1)}, Z^{(1)}, R_0^{(1)}, R_1^{(1)}, ..., R_5^{(1)} \in QR_{n_1}, n_1)$ and $(S^{(2)}, Z^{(2)}, R_0^{(2)}, R_1^{(2)}, ..., R_5^{(2)} \in QR_{n_2}, n_2)$.

6.2.1 Design and implementation

In the case of standard and domain pseudonyms, the associated pseudorandomness to r and m_0 must be recomputed by the PRNG that we presented in Section 4. Therefore, the PRNG would follow the $init_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow \tilde{r} \Rightarrow \tilde{m}_0 \Rightarrow r \Rightarrow reset_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow \tilde{r} \Rightarrow \tilde{m}_0$ sequence in order to recompute the required pseudorandom values during the generation of both *t-* and *s_values* in the case of proving the ownership of a standard pseudonym and a domain pseudonym.

Any strategy for computing equality proofs of representation must take into account that we need to main in RAM two or more (A, e, v) tuples in order to generate each *t_value* and maintain a reasonable performance figure. Besides a hash function is needed for computing a challenge c that relies on multiple blocks of data (*t_values*) and common values. The MULTOS hash function for obtaining a SHA-256 digest requires the full input in memory, and that resource is limited in our target device, we must find an alternative function that can compute hashes with partial inputs in a subsequent manner.

We combined the PRNG we presented in Section 5 with variable reconstruction in RAM. First The $(\hat{e}, \hat{v}')$ values only depend on the $(\tilde{e}, \tilde{v})$ pseudorandom variables. They do not depend on $\tilde{m}$, the same space reserved in RAM for such value (74 bytes as described in Section 5) can be reused for $(\hat{e}, \hat{v}')$ if their size is adapted to the largest value (i.e. 255 bytes in the case of $\tilde{v}$). This approach, makes it possible to sequentially reconstruct via PRNG $\hat{e}$ and $\hat{v}'$ (i.e. as $\hat{e} = \tilde{e} + ce'$ and $\hat{v}' = \tilde{v} + cv'$) during the generation of the *t-* and *s_values*. Second, the randomized computation of the signature component A' requires r_A (it can be derived via PRNG). Since the randomization of this value is independent from (e, v) and $\hat{m}_i$ values, we can compute all these variables in a sequentially. After each pseudorandom value has been recomputed, the reconstructed variable is temporary stored in the transaction memory of the card till it is requested by the veri-

[11] In this case $\alpha_0 = \beta_0$ if both credentials belongs to the same cardholder. We represent the non-disclosed master secret as μ following the Camenisch-Staedler notation [16].

fier. Thus, the PRNG sequence in this case whould be $init_{PRNG}() \Rightarrow r_A^{(1)} \Rightarrow \tilde{v}^{(1)} \Rightarrow \tilde{e}^{(1)} \Rightarrow \tilde{m}_i^{(1)} \Rightarrow r_A^{(2)} \Rightarrow \tilde{v}^{(2)} \Rightarrow \tilde{e}^{(2)} \Rightarrow \tilde{m}_i^{(2)} \Rightarrow reset_{PRNG}() \Rightarrow r_A^{(1)} \Rightarrow \tilde{v}^{(1)} \Rightarrow \tilde{e}^{(1)} \Rightarrow \tilde{m}_i^{(1)} \Rightarrow r_A^{(2)} \Rightarrow \tilde{v}^{(2)} \Rightarrow \tilde{e}^{(2)} \Rightarrow \tilde{m}_i^{(2)}$.

Table 6 shows the performance figures for recomputing the pseudorandomness related to $(A', \hat{v}', \hat{e}, \hat{m}_i)$. The number of calls to the PRNG is higher in the case of $\hat{v}'$ but the overall execution time is preponderant during the reconstruction of A' since it requires recomputing the S^{r_A} modular exponentiation (235.191 ms). In contrast to the execution model described in Section 3.2, we have rearranged the computation of $(A', \hat{e}, \hat{v}')$ to $(A', \hat{v}', \hat{e})$ since the computation of $\hat{v}'$ requires r_A. On the contrary, $\hat{e}$ does not depend on other values.

Table 5 Time required for reconstructing A'_i, $\hat{v}'_i$, $\hat{e}_i$, and $\hat{m}_i$ in RAM

Variable	Operation	Size (bytes)	Calls to PRNG	Delay (ms)
A'_i	AS^{r_A}	138	9	235.191
$\hat{v}'_i$	$v' = v - e \cdot r_A$ $\hat{v} = \tilde{v}' + c \cdot v'$	255	16	104.365
$\hat{e}_i$	$\tilde{e} + c \cdot e'$	57	4	30.710
$\hat{m}_i$	$\tilde{m}_i + c \cdot m_i$	74	5	36.708

The MULTOS primitive `PRIM_SECURE_HASH_IV` is used for hashing each *t_value* in steps so we do not need to maintain a considerable amount of bytes in RAM in order to generate the challenge associated to the proof. Once we have added the last *t_value* to the hash computation, the transaction nonce is hashed and the final digest is derived. In contrast to Eq. 2, the *s_values* for each credential signature $(\hat{e}, \hat{v}')$ are now recomputed on demand when the verifier request them. Consequently, that latency is added to $T_{get\_sig}(i)$.

Finally, we can do additional optimizations if instead of considering equality proofs over n credentials we restrict ourselves to a pair of them. We can store both $A'^{(1)}$ and $A'^{(2)}$ in transient memory and avoid recomputing them two times as described in Table 6. Moreover, $r_A^{(1)}$ and $r_A^{(2)}$ can be stored too in order to avoid regenerate them via the PRNG. Given that the transient memory of our target device has a size of 1,016 bytes and the APDU buffer is limited to 256 bytes according to the ISO 7816 standard we can use up to 1,016 - 256 = 760 bytes for storing these values[12]. Hence the PRNG sequence for this approach is represented by $init_{PRNG}() \Rightarrow r_A^{(1)} \Rightarrow \tilde{v}^{(1)} \Rightarrow \tilde{e}^{(1)} \Rightarrow \tilde{m}_i^{(1)} \Rightarrow r_A^{(2)} \Rightarrow \tilde{v}^{(2)} \Rightarrow \tilde{e}^{(2)} \Rightarrow \tilde{m}_i^{(2)} \Rightarrow reset_{PRNG}() \Rightarrow \tilde{v}^{(1)} \Rightarrow \tilde{e}^{(1)} \Rightarrow \tilde{m}_i^{(1)} \Rightarrow \tilde{v}^{(2)} \Rightarrow \tilde{e}^{(2)} \Rightarrow \tilde{m}_i^{(2)}$ skipping the regenerated values $r_A^{(1)}, r_A^{(2)}$ (stored).

[12] We need $2 \cdot 128$ bytes for $A'^{(1)}$, $A'^{(2)}$ and $2 \cdot 138$ bytes for $r_A^{(1)}$ and $r_A^{(2)}$

6.2.2 Results

We have depicted in Table 4 the performance figures of the three primitives described in this section. We must note that T_{get_attr} is related to $T_{get\_attr}(m_0, ..., m_5, nym, \hat{r}|dNym)$ when computing standard and standard and domain pseudonyms combined whereas it is related to $T_{get\_attr}(m_0, ..., m_5)$ in the case of the equality proofs. This is also the case of the selective disclosure operation.

We have also represented the RA/HA case scenarios for each type of pseudonym. In this case, since we need to perform an extra number of modular exponentiations related to the pseudonyms commitments (e.g. nym), this proof required 1,176.02 - 840.73 = 335.29 ms in the best case. Besides, we have compared the first approach for computing n proofs of equality with the optimizations carried out for computing only 2, that is, using two different strategies for 2 credentials. Relying on the optimizations for 2 credentials, it can be possible to perform an equality proof of representation in 2,261.19 ms revealing all the attributes, whereas hiding all the attributes would require 1,129.40 extra ms. This is possible since we do not need to recompute $(\hat{e}, \hat{v}')$ for each credential in the generation of each *t_value*, increases the execution time of $T_{gen\_commit}(n, r)$. The use case of two credentials can be interesting where one of the credentials e.g. root, is related to a certain organization and contains only data related to it. A second credential, containing information about the cardholder where their identity can be partially disclosed can be used in combination with the root one in order to proof his membership in different scenarios.

7 Performance evaluation of complex proofs via prime-encoded attributes

In the prior section we have analyzed the performance of different operations over a credential based on attributes represented as strings of a certain length. In this case, the number of modular exponentiations when proving the ownership of attributes is linear with the credential size. On the other hand, Camenisch et al. proposed to encode each attribute as a prime number in order to the reduce the number of modular exponentiations to 2 with independence of the credential size [9]. This makes it possible to selective disclose attributes using the co-prime property (AND operator), to prove the absence of an attribute in a credential (NOT operator) and to prove that one or more attributes can be present (OR operator). The only based utilized e.g. R_1 encodes all the credential attributes as a product $m_t = \prod_{i=1}^{l} m_i$ for l attributes.

We rely on the PRNG we described in Section 5 for making it possible the execution of these proofs and operators and introduce two techniques (internal and external commitment reorganizations) for chaining the three operators.

7.0.1 Issuing prime-encoded attributes

First, we analyze the issuing operations of prime-encoded attributes and compare it with the current implementation of the IRMA card [29]. It is expected that issuing prime-encoded attributes could reduce the latency associated to $T_{issuing}(n)$ as the number of attributes increase (Section 4). In the IRMA card, only 5 attributes w.r.t. the bases $R_0, ..., R_5$ are used. On the other hand, we can store any number of prime-encoded attributes s.t. the only limitation would be the prime size.

Since we wanted to know what is the maximum number of attributes per credential that we can issue following this scheme we relied on the following methodology. We set a limit of 50 attributes per credential restricted in size so $|m_t|$ cannot be greater than the l_m = 256 bit limitation according to the Idemix specification. Then, we created the following cases: (1) one possibility per attribute: we rely on the first 50 primes, (2) 10 possibilities per attribute: we rely on the first 500 primes, (3) 100 possibilities per attribute, we rely on the first 5,000 primes, (4) 1,000 possibilities per attribute, we rely on the first 50,000 primes. We select attributes from the list of the first 50 primes, 500 primes, 5,000 primes, 50,000 primes and so on in order to construct our credentials w.r.t. the m_t exponent for the base R_1 as $\prod_{i=0}^{l} m_i$ for $l = 50 - 1$.

We can randomly choose prime numbers from that list and construct our credentials from 1 to 50 attributes, stopping when $|m_t| \geq 256$ bits. After repeating this experiment 100 times for each case, we obtained the approximate maximum number of attributes: 44 attributes (case 1), 25 attributes (case 2), 18 attributes (case 3) and 14 attributes (case 4) as depicted in Fig. 1.

As showed, it is possible to not cross the 3 seconds performance of the IRMA card for issuing 5 attributes and at the same time issue 44, 22, 18 and 14 attributes of different lengths with a performance under 2.7 seconds. This means that it would be possible to combine the issuing operation with another proof such as the issuance of a pseudonym of several, maintaining at the same time a similar performance of the IRMA but, in contrast, issuing from 3 to 8 times more attributes [29].

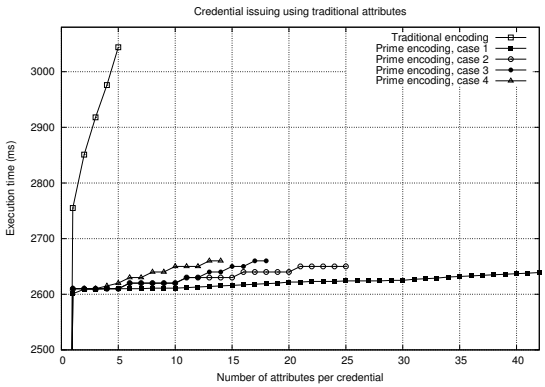

Fig. 1 Performance of issuing attributes via prime-encoded and traditional attributes

7.1 The AND operator

This operator enable the cardholder to selective disclosure attributes proving that an attribute m_i divides m_t. This is represented in zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, M\alpha_1) : Z \equiv \pm R_0^{\alpha_0}(R_1^{m_1})^{\alpha_1} A^{\varepsilon'} S^{\nu'} \mod n\}$. In addition, the commitments $C = Z^{m_t} S^r \mod n$, $\tilde{C}$ =M $(Z^{m_1})^{\tilde{m_h}} S^r M \mod n$ and $\tilde{C_0} = Z^{\tilde{m_t}} S^{\tilde{r}} \mod n$ must beM computed, where $m_h = Mm_t/m_i$ and m_i consists of the product of attributes that areM revealed (in this case $m_i = m_1$). The PRNG computes the following sequence: $init_{PRNG}() \Rightarrow \tilde{m_i} \Rightarrow \tilde{m_h} \Rightarrow \tilde{r} \Rightarrow r \Rightarrow \tilde{m_t} \Rightarrow reset_{PRNG}() \Rightarrow \tilde{m_i} \Rightarrow \tilde{m_h} \Rightarrow \tilde{r}$. Not revealing any attribute requires two exponentiations with independence of the number of attributes hidden.

7.1.1 Internal reorganization of commitments

In the computation of the AND proofs we need to commit to the m_t value, i.e. the first attribute of the first base as $C = Z^{m_t} S^r \mod n$. However, the next commitment requires the computation of the S^r again as $\tilde{C} = (Z^{m_1})^{\tilde{m_h}}) S^r$. In order to avoid recomputing S^r, we can proceed by reordering all the computations and reuse this value from the last commitment. In this case, the order of computations would be (1) $Z^{m_t} S^r$, (2) $[S^r](Z^{m_1})^{\tilde{m_h}}$ by leaving the result S^r in RAM and proceeding with the next multiplication. This resulted in an speed up of 78 ms per operation.

7.1.2 Selective disclosure

Due to the computation of the C, $\tilde{C_o}$ and $\tilde{C}$ commitments together with the two extra response values, revealing attributes via the AND operator undermines any speed up in comparison to the issuing operation relying on traditional attributes. As depicted in Figure 2 we notice that what we considered the worst case

for normal encoding is the opposite here. That is, hiding attributes is computationally more expensive using traditional attributes in comparison to prime-encoded ones whereas hiding attributes only has a constant performance related to prove the ownership of m_0 and m_t w.r.t. $R_0^{m_0} R_1^{m_t}$.

Besides, the cost of this operation is related to the computation of the Z^{m_r} exponentiation w.r.t. of m_r as the product of the cardholder's attributes that are revealed together with the product itself (computed during $T_{gen\_commit}(n, r)$, Eq. 3). Therefore, it is expected that the AND operator increases the computation time as the number of attributes are revealed at a speed related to the primes utilized (Figure 2). In this respect, an alternative for reducing the latency of $T_{gen\_commit}(n, r)$ is to precompute a restricted set of combinations for revealing attributes Z^{m_r} and store them in ROM so $T_{gen\_commit}(n, r)$ is constant w.r.t. $C, C_0, \tilde{C}$.

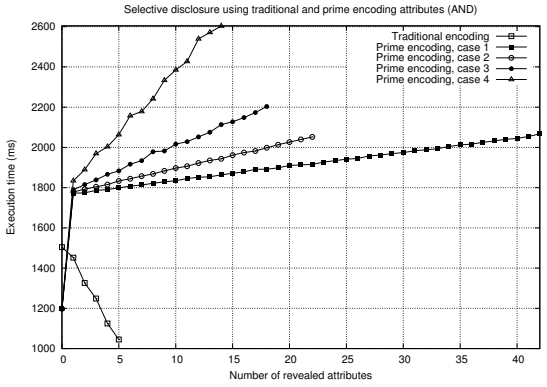

Fig. 2 Performance of selective disclosure using both types of encoding

7.1.3 Standard and domain pseudonyms

In cases where hiding attributes and therefore, maintain the identity of the cardholder private is possible when proving the ownership of pseudonyms we can rely on prime-encoded attributes.

7.1.4 Equality proofs of representation

We can obtain better performance figures of equality proofs of representation than those obtained in Section 6.2.2 using prime-encoded attributes when we hide them. Proving that 2 credentials share m_0 without revealing any attributes would be represented in zero knowledge as NIZK: $\{(\varepsilon_1', \nu_1', \varepsilon_2', \nu_2', \mu, \alpha_1, \alpha_2) : Z^{(1)} \equiv \pm R_0^{(1)\mu} R_1^{(1)\alpha_1} A^{(1)\varepsilon_1'} S^{(1)\nu_1'} \mod n_1 \wedge Z^{(2)} \equiv \pm R_0^{(2)\mu} R_1^{(2)\beta_1} A^{(2)\varepsilon_2'} S^{(2)\nu_2'} \mod n_2\}$.

However, the computation of C, $\tilde{C}_o$ and $\tilde{C}$ together with the two extra *s_values* undermines any possibility of improving the results from Section 6.2.2.

We have estimated the time that requires computing equality proofs of up to 8 credentials using the three alternatives we have presented so far. Within a margin of 4–5 seconds. Considering 4–5 seconds the acceptable time for an on-line setting. Hence, performing equality proofs with 3 and 4 credentials revealing all the attributes would be possible whereas execution times beyond 6 seconds (worst cases with 3 credentials and beyond and best cases with 5 credentials and beyond) are unrealistic in practical scenarios.

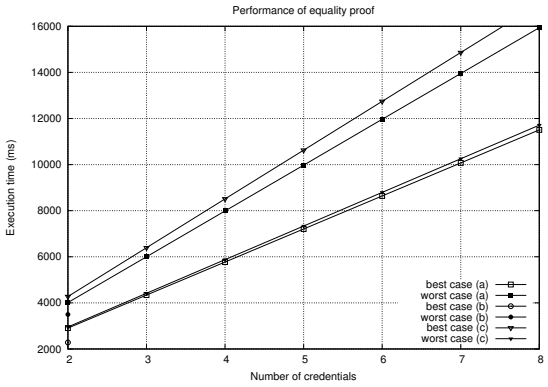

Fig. 3 Performance of the equality proof (credentials of five attributes)

Finally, we notice that it is possible to improve the performance results of an equality proof of 2 credentials hiding all the attributes by using this type of encoding. This approach would be only useful in systems where a user should prove the ownership of n credentials without revealing her attributes.

7.2 The NOT operator

A cardholder can prove that a certain attribute or group of attributes (m_i) are not in her credential via this operator s.t. they do not belong to m_t. In so doing, we proof in zero knowledge that two integers x, y exists w.r.t. the following linear Diophantine equation $x \cdot m_t + y \cdot m_i = 1$: NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_1, \chi, \upsilon, \rho, \rho') : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} A^{\varepsilon'} S^{\nu'} \mod n \wedge C \equiv \pm Z^{\alpha_1} S^{\rho} \mod n \wedge Z \equiv \pm C^{\chi} (Z^{m_i})^{\upsilon} S^{\rho'}\}$ mod n. The card must compute the commitments $C = Z^{m_t} S^{r} \mod n$, $\tilde{C} = C^{\tilde{x}} (Z^{m_i})^{\tilde{y}} S^{\tilde{r}'} \mod n$ and $\tilde{C}_c = Z^{\tilde{m}_t} S^{\tilde{r}} \mod n$ where $\tilde{r}$, $\tilde{r}'$, $\tilde{x}$,$\tilde{y}$ are randomizers [9].

We propose two implementations for computing the (x, y) pairs. The first alternative consists of precomputing them. The equation (x, y) for $x \cdot m_i + y \cdot m_t = 1$, m_t has always the same value and there are several

Table 6 Performance of GCD using the proposed algorithms

Case	$\|m_i\|$ (bytes)	$\|m_t\|$ (bytes)	Euclid (ms)	Stein (ms)	Lehmer (ms)	Extended Euclidean Algorithm (Euclid, ms)
1	1	2	**17.05**	70.62	18.43	**21.51**
2	1	4	**17.52**	131.78	18.92	**21.76**
3	3	6	**37.49**	254.37	38.86	**65.64**
4	5	9	**68.07**	289.58	95.61	**131.47**

combinations for m_i. We can store every possibilities in EEPROM when using a small number of attributes. The number of (x, y) pairs is related to the number of attributes as $\sum_{i=1}^{l} c_n^i = \sum_{i=1}^{l} \binom{n}{i}$. For instance, for $l = 2$ attributes per credential, we store three (x, y) pairs and for $l = 4$, we store 15 pairs. The second approach has to do with the computation of the Extended Euclidean algorithm on the smart card. We extracted performance figures via the binary GCD or Stein's algorithm, Lehmer's algorithm and an implementation of the Extended Euclidean algorithm via the `PRIMDIVIDEN` [13] instruction of the MULTOS card [25].

The length of m_i varies according to the number of attributes that are proved to not belong to m_t. We measured the performance figures of these algorithms via 4 cases[14]. These cases assumes credentials of 5 attributes so we can compare our results with those of the IRMA card [29].

As depicted in Table 6, the Stein's variant did not improve the performance figures of the implementation relying on the traditional Extended Euclidean algorithm. Hence, replacing the arithmetic operations of the traditional Extended Euclidean algorithm by bitwise operations did not improve our performance figures. We measured the latency of all the operations involved in both algorithms: Euclidean division (11.852 ms), comparison (11.047 ms), Boolean and (10.411 ms), right shift (10.634 ms), increment (10.354 ms) and subtraction (10.647 ms). That means that these latency's make any improvement ill-suited when replacing the Euclidean division by other type of operations. Moreover, due to the proprietary nature of the the SLE 78CX1280P chip we cannot claim that the Euclidean division is being performed via the hardware accelerator of the target device. In the case of the Lehmer's variant, for single-precision values we obtain similar results as the Euclidean algorithm. We believe that due to that when we overcome that value (multi-precision), there are more calls to the operating system for performing bit-wise operations, multiplications and divisions that increase the latency of the algorithm despite this is not expected, whereas in the traditional Euclidean algorithm we are only performing one Euclidean division by step.

[13] This instruction computes the Euclidean division of two numbers i.e. q and r.

[14] Thus, for one possibility per attribute, we prove the non-existence of one attribute in m_i. In this case, $m_i = 3$ and $m_t = 5 \cdot 7 \cdot 11 \cdot 13$ (case 1). We consider 10 possibilities per attribute (50 primes). We prove the non-existence of one attribute in m_i. For $m_i = 3$, $m_t = 179 \cdot 181 \cdot 191 \cdot 193$ (case 2). We consider 1,000 possibilities per attribute (i.e. 5,000 primes) and we prove the non-existence of two attributes in m_t for $m_i = 1,999 \cdot 2,161$ and $m_t = 3,323 \cdot 3,253 \cdot 2,897 \cdot 2,999$ (case 3). Finally, we consider 10,000 possibilities per attribute (50,000 primes) and we proof the non-existence of two primes $m_i = 91,387 \cdot 91,393$ in $m_t = 102,461 \cdot 102,481 \cdot 102,497 \cdot 102,499$ (case 4).

7.3 The OR operator

A verifier could send a list of attributes to the card so the cardholder could prove that one or more attributes, encoded as a product can be found in m_t s.t. $m_t = \prod_{i=0}^{l}$ w.r.t. $R_1^{m_t}$. Given an attribute $m_i \in m_t$, an integer x exists s.t. $x \cdot m_i = \prod_{i=1}^{l} m_i = m_t$. We represent this in zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_1, \chi, \rho, \rho') : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} A^{\varepsilon'} S^{\nu'} \mod n \wedge C \equiv \pm Z^{\alpha_1} S^{\rho} \mod n \wedge C' \equiv \pm C^{\chi} S^{\rho'}\} \mod n$[15]. The card must compute three commitments $C = Z^{m_t} \cdot S^{r} \mod n, \tilde{C} = Z^{\tilde{m_t} \cdot S^{\tilde{r}}} \mod n, \tilde{T} = C^{\tilde{x}} \cdot S^{\tilde{r_1}}$ w.r.t $x = \frac{m_t}{m_i}$ s.t. $R_1^{m_t}$ and $r_1 = -r_0 \cdot x$ where $r, r_0, \tilde{r}, \tilde{r_0}, \tilde{r_1}, \tilde{m_t}, \tilde{x}$ are randomizers.

The computations of parameters such as r_1 require signed arithmetic that is not supported in the card. That mean that we had to wrap every multiplication and addition for supporting sign extensions and two's complement arithmetic. By using the RAM reductions achieved thanks to the PRNG described in Section 5 by executing all the two's complement operations in RAM, we cold reduce the computational time of $r_1 = -\rho_0 \cdot \chi$ from 495.530 ms to 90.260 ms.

7.4 Results

We have represented in Table 7 our performance figures of measured proofs over prime-encoded credentials.

[15] In this manuscript we only address the first version of this NIZK described in [9] and leave the second one beyond the scope of this work due the computation limitations of our target device.

In this case, T_{get_attr} is $T_{get\_attr}(\hat{m_0}, \hat{m_t}|\hat{r}, \hat{r}', \hat{a}, \hat{b}, C)$ for the NOT and OR proofs. Besides, $T_{get\_attr}(m_0, m_1|C, \hat{r}, \hat{m_h})$ is for the AND operator where performing the selective disclosure of attributes. For pseudonyms is related to $T_{get\_attr}(m_0, m_1|C, \hat{r}, \hat{m_h}|nym, \hat{r}|dNym)$ and $T_{get\_attr}(m_0, m_t|C, \hat{r}, M\hat{m_h})$ for equality proofs of representation.

By performing selective disclosure via the AND operator, only proving the ownership of a CL signature over a set of attributes requires 1,198.21 ms. However, to reveal every attribute with the exception of the master secret requires the computation of C, $\tilde{C}$ and $\tilde{C_0}$. We can store C in RAM thanks to the optimization described in section 5. All in all, revealing all the attributes requires 492.20 ms more in comparison to the utilization of traditional encoding due to the computation of the of C, $\tilde{C}$ and $\tilde{C_0}$ commitments.

When computing pseudonyms via prime-encoded proofs all the performance figures concerning the worst cases (i.e. hiding all, HA) were improved i.e. 485.89 ms (standard pseudonyms) and 499.76 ms (domain pseudonyms in combination with standard pseudonyms, Section 6). However, revealing all the attributes requires the computation of three commitments that need a larger number of modular arithmetic operations in comparison to normal encoding as we have seen. The same applies to computing equality proofs of representation where there is an improvement of 496.59 ms and 989.82 ms respectively in the cases where all the attributes are hidden.

In the case of the NOT operator[16], the precomputation strategy shows that increasing the length of the operand does not alter the result significantly. In contrast, the time required for obtaining the (x, y) pairs during the computation of $\hat{a}$ sums up to $T_{get\_attr}(i)$. Finally, thanks to the MULTOS PRIMDIVIDEN operation, we obtained performance figures between 2,015.41 and 2,135.35 ms. Finally, we can compute the OR operator in 1,885.96 ms for credentials of 5 attributes.

7.5 Combination of operators for prime-encoded credentials

By combining AND, OR and NOT proofs a cardholder could proof that for attributes (a, b, c, d), a is in m_t s.t. $R_1^{m_t}$, b is not and c or d could be present. However, given that the respective proofs for this operators require the same commitments, it is possible to reorganize the products inside them (thus doing an *external* reorganization of them) in order to optimize the overall performance figure of the combined operators:

1. AND $\wedge$ NOT: In the AND proof we always to commit to m_t as $C = Z^{m_t} \cdot S^r$ in order to prove that a certain m_1 can divide m_t afterward and utilize the $\tilde{m_t}, \tilde{r}$ randomizers for proving the ownership of m_t as $\tilde{C_0} = Z^{\tilde{m_t}} \cdot S^{\tilde{r}}$. The response values $\hat{m_t}, \hat{r}$ are created. The NOT operator follows a similar approach for proving the ownership of m_t in the case of the C and $\tilde{C_c}$ commitments (Section 7.2). Hence, when proving both presence an absence of attributes one can avoid computing these two commitments and their response values twice. Moreover, in the case of AND we can apply internal commitment reorganization as discussed in Section 7.1.1.
2. AND $\wedge$ OR: The OR operator (Section 7.3) proves the ownership of m_t as $C = Z^{m_t} S^r$ and generates $\tilde{C}$ as the AND and NOT operator as well as the response values for $\hat{m_t}, \hat{r}$. This means that it can be computed only e one time when combined and the AND proof can be executed with the optimizations discussed in Section 7.11.
3. NOT $\wedge$ OR: As discussed, both operators compute the $C, \tilde{C}$ commitments and only need to be obtained once. However, none of these operators enable the possibility of performing internal commitment reorganizations.
4. AND $\wedge$ NOT $\wedge$ OR: Finally, this is the combination that enable us to perform a greater number of optimizations, First, $C, \tilde{C}, \hat{m_t}, \hat{r}$ do not need to be performed three times and we can optimize the proof related to the AND operator as described in Section 7.1.1.

We have depicted in Table 9 the time reductions we obtained while performing external and internal commitment reorganizations for credentials of 5 attributes. In so doing, we obtained reductions in performance that varies between 170.10 ms and 644.60 ms.

8 pIRMA

For extracting our performance figures, we developed an open-source implementation of Idemix in Python, pIRMA[17]. It implements the client-side, that is, the verifier part of Idemix and enabled us to extract performance figures of the smart card as well to develop Idemix clients and related applications. We have released this implementation as GPL together with the respective MULTOS code.

[16] We use the following notation: PRE means recomputing, EUC 1-3 is related to the cases presented in Table 3, RA means Reveal all the Attributes with the exception of the master secret and HA to hide every attribute in the credential.

[17] `https://github.com/adelapie/irma_phase_2`

Table 7 Performance figures of complex proofs via prime-encoded attributes

Implementation	Case	T_{sel_cred}	T_{gen_commit}	$T_{get\_sig}(A, e, v)$	T_{get_attr}	Total (ms)
Selective disclosure (AND)	HA	103.12	1,071.1	46.72	65.17	**1,286.1**
Selective disclosure (AND)	RA	103.27	1,562.4	46.87	178.30	1,890.8
Nym	RA	103.13	1,720.73	46.10	213.06	2,083.01
Nym	HA	104.11	1,288.88	46.40	141.07	**1,579.53**
Nym $\wedge$ dNym	RA	103.17	2,255.31	45.92	272.50	2,676.92
Nym $\wedge$ dNym	HA	104.33	1,487.86	46.19	199.85	**1,838.25**
Equality proof (2 cred.)	RA	103.23	3,145.11	703.56	253.01	4,202.74
Equality proof (2 cred.)	HA	104.17	2,023.94	703.50	124.76	**2,894.01**
NOT	PRE	15.203	1,590.11	48.72	214.80	**1,974.96**
NOT	EUC (1)	15.503	1,587.93	46.81	259.95	2,016.41
NOT	EUC (2)	15.514	1,587.92	47.10	260.53	2,017.23
NOT	EUC (3)	15.510	1,587.93	46.87	306.28	2,063.63
NOT	EUC (4)	15.511	1,587.91	46.85	376.28	2,135.35
OR	-	113.622	1,476.47	46.08	234.53	1,885.96

Table 8 Estimation of the performance obtained by the combination of operators for prime-encoded credentials (5 attributes)

Combination	Case	Performance (ms)	Performance after optimization (ms)
AND $\wedge$ NOT	RA, PRE	2,485.3	2,201.9
AND $\wedge$ NOT	RA, EUC1	2,506.9	2,223.4
AND $\wedge$ NOT	RA, EUC2	2,507.1	2,223.7
AND $\wedge$ NOT	RA, EUC3	2,551.0	2,267.5
AND $\wedge$ NOT	RA, EUC4	2,616.8	2,333.4
AND $\wedge$ OR	RA, C1	2,247.7	1,924.5
NOT $\wedge$ OR	PRE, C1	2,292.3	2,122.2
NOT $\wedge$ OR	EUC1, C1	2,397.9	2,227.8
NOT $\wedge$ OR	EUC2, C1	2,365.2	2,195.1
NOT $\wedge$ OR	EUC3, C1	2,409.1	2,238.9
NOT $\wedge$ OR	EUC4, C1	2,474.9	2,304.8
AND $\wedge$ NOT $\wedge$ OR	RA, PRE, C1	2,897.1	2,252.6
AND $\wedge$ NOT $\wedge$ OR	RA, EUC1, C1	2,918.6	2,274.1
AND $\wedge$ NOT $\wedge$ OR	RA, EUC2, C1	2,918.9	2,274.3
AND $\wedge$ NOT $\wedge$ OR	RA, EUC3, C1	2,962.8	2,318.2
AND $\wedge$ NOT $\wedge$ OR	RA, EUC4, C1	3,028.6	2,384.0

9 Conclusions

The Idemix private ABC system consists of a variety of primitives such as the selective disclosure of attributes, equality proofs, Idemix pseudonyms, domain pseudonyms, and the AND, OR, NOT operators on prime-encoded attributes. In this manuscript, we have evaluated the performance of all these operations on a MULTOS card, relying on the IRMA implementation, an open-source implementation of the issuing and selective disclosure operations of Idemix for credentials of 5 attributes. This means that we have obtained performance figures of the 77% of the Idemix specification, while in the literature several practitioners restricted themselves to the selective disclosure of attributes in credentials of one attribute [4, 28].

Further, by relying on a PRNG and a small amount of RAM available on card, we were able to extract the performance figures of the operation describes. In Table 9, we have depicted the savings in RAM performing variable reconstruction and the internal and external reorganization of commitments. We note the size of the variable in RAM that we are using for storing the required pseudorandomness in each case. For instance, during the execution of an equality proof since the greater value we have to regenerate is r_A, this container has a size of 255 byte with independence of the randomizer that is being generated sequentially. In the optimized version for performing equality proofs for 2 credentials, since that value is stored in the transient memory, the size of the container is 138 bytes. We also must note that when the HA case of the AND operator is used in combination with other operators, only the commitments related to the other ones are taken into account since the only aspect related to this proof is

Table 9 RAM savings by recomputing the pseudorandomnes in each primitive

Operator	PRNG sequence	No. of commitments	Required RAM (bytes)	RAM saved (bytes)
Selective disclosure RA	$\tilde{m}_i \vert \tilde{m}_i$	1	$1 \cdot 74$	-
Selective disclosure HA	$\tilde{m}_i \vert \tilde{m}_i$	1	$5 \cdot 74$	$4 \cdot 74$
Nym RA	$\tilde{m}_i, \tilde{r}, \tilde{m_0}, r \vert \tilde{m}_i, \tilde{r}, \tilde{m_0}$	2	$1 \cdot 74 + 1 \cdot 74$	$1 \cdot 74$
Nym HA	$\tilde{m}_i, \tilde{r}, \tilde{m_0}, r \vert \tilde{m}_i, \tilde{r}, \tilde{m_0}$	2	$5 \cdot 74 + 1 \cdot 74$	$4 \cdot 74 + 1 \cdot 74$
Nym ∧ dNym RA	$\tilde{m}_i, \tilde{r}, \tilde{m_0}, r \vert \tilde{m}_i, \tilde{r}, \tilde{m_0}$	3	$1 \cdot 74 + 1 \cdot 74$	$1 \cdot 74$
Nym ∧ dNym HA	$\tilde{m}_i, \tilde{r}, \tilde{m_0}, r \vert \tilde{m}_i, \tilde{r}, \tilde{m_0}$	3	$5 \cdot 74 + 1 \cdot 74$	$4 \cdot 74 + 1 \cdot 74$
Eq. proof (n cred.) RA	$r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)} \vert r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)}$	2	$2 \cdot 138 + 2 \cdot 255 + 2 \cdot 57 + 2 \cdot 74$	$2 \cdot 138 + 1 \cdot 255 + 2 \cdot 57 + 2 \cdot 74$
Eq. proof (n cred.) HA	$r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)} \vert r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)}$	2	$2 \cdot 138 + 2 \cdot 255 + 2 \cdot 57 + 10 \cdot 74$	$2 \cdot 138 + 1 \cdot 255 + 2 \cdot 57 + 10 \cdot 74$
Eq. proof (2 cred.) RA	$r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)} \vert \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)}$	2	$2 \cdot 138 + 2 \cdot 57 + 2 \cdot 74$	$1 \cdot 138 + 2 \cdot 57 + 2 \cdot 74$
Eq. proof (2 cred.) HA	$r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)} \vert \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)}$	2	$2 \cdot 138 + 2 \cdot 255 + 2 \cdot 57 + 10 \cdot 74$	$1 \cdot 138 + 2 \cdot 57 + 10 \cdot 74$
Eq. proof (prime-encoded) RA	$r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m_0}, \tilde{m_h}, \tilde{r}, r, \tilde{m_t}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)} \vert [\_], \tilde{m_0}, \tilde{m_h}, \tilde{r}, [\_], [\_], \tilde{m_t}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m}_i^{(1)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m}_i^{(2)}$	5	$2 \cdot 138 + 2 \cdot 57 + 5 \cdot 74$	$1 \cdot 138 + 2 \cdot 57 + 5 \cdot 74$
Eq. proof (prime-encoded) HA	$r_A^{(1)}, \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m_t}, r_A^{(2)}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m_0} \vert \tilde{v}^{(1)}, \tilde{e}^{(1)}, \tilde{m_t}, \tilde{v}^{(2)}, \tilde{e}^{(2)}, \tilde{m_0}$	2	$2 \cdot 138 + 2 \cdot 255 + 2 \cdot 57 + 2 \cdot 74$	$1 \cdot 138 + 2 \cdot 57 + 2 \cdot 74$
AND RA	$\tilde{m_0}, \tilde{m_h}, \tilde{r}, r, \tilde{m_t} \vert [\_], \tilde{m_0}, \tilde{m_h}, \tilde{r}, [\_], [\_], \tilde{m_t}$	4	$3 \cdot 74$	$2 \cdot 74$
AND HA	$\tilde{m_t}, \tilde{m_0} \vert \tilde{m_t}, \tilde{m_0}$	1	$1 \cdot 74$	$1 \cdot 74$
AND ∧ Nym RA	$\tilde{m_0}, \tilde{m_h}, \tilde{r}, r', r, \tilde{m_t} \vert [\_], \tilde{m_0}, \tilde{m_h}, r', \tilde{r}, [\_], [\_], \tilde{m_t}$	5	$5 \cdot 74$	$4 \cdot 74$
AND ∧ Nym HA	$\tilde{m_t}, \tilde{m_0}, r' \vert \tilde{m_t}, \tilde{m_0}, r'$	2	$2 \cdot 74$	$1 \cdot 74$
NOT	$\tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}, r \vert \tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}$	4	$6 \cdot 74$	$5 \cdot 74$
OR	$r, \tilde{m_t}, \tilde{r}, \tilde{r_1}, \tilde{x}, \tilde{m_0} \vert [\_], \tilde{m_t}, \tilde{r}, \tilde{r_1}, \tilde{x}, \tilde{m_0}$	3	$5 \cdot 74$	$4 \cdot 74$
AND RA ∧ NOT	$\tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}, r, \tilde{m_h} \vert \tilde{b}, \tilde{a}, [\_], \tilde{m_t}, \tilde{m_0}, [\_], \tilde{m_h}$	6	$5 \cdot 74$	$4 \cdot 74$
AND HA ∧ NOT	$\tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0} \vert \tilde{b}, \tilde{a}, [\_], \tilde{m_t} [\_], \tilde{m_0}$	5	$4 \cdot 74$	$3 \cdot 74$
AND RA ∧ OR	$\tilde{m_0}, \tilde{m_h}, \tilde{r}, r, \tilde{m_t}, \tilde{r_1}, \tilde{x} \vert \tilde{m_0}, \tilde{m_h}, \tilde{r}, [\_], \tilde{m_t}, \tilde{r_1}, \tilde{x}$	5	$6 \cdot 74$	$5 \cdot 74$
AND HA ∧ OR	$r, \tilde{m_t}, \tilde{r}, \tilde{r_1}, \tilde{x}, \tilde{m_0} \vert [\_], \tilde{m_t}, \tilde{r}, \tilde{r_1}, \tilde{x}, \tilde{m_0}$	3	$5 \cdot 74$	$4 \cdot 74$
NOT ∧ OR	$\tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}, r', \tilde{r}, \tilde{r_1}, \tilde{x} \vert \tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}, \tilde{r}', [\_], \tilde{r_1}, \tilde{x}$	5	$9 \cdot 74$	$8 \cdot 74$
AND RA ∧ NOT ∧ OR	$\tilde{m_0}, \tilde{m_h}, r, r', \tilde{m_t}, \tilde{b}, \tilde{a}, \tilde{r_1}, \tilde{x} \vert \tilde{m_0}, \tilde{m_h}, [\_], [\_], \tilde{m_t}, \tilde{b}, \tilde{a}, \tilde{r_1}, \tilde{x}$	5	$10 \cdot 74$	$9 \cdot 74$
AND HA ∧ NOT ∧ OR	$\tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}, r', \tilde{r}, \tilde{r_1}, \tilde{x} \vert \tilde{b}, \tilde{a}, \tilde{r}', \tilde{m_t}, \tilde{r}, \tilde{m_0}, \tilde{r}', [\_], \tilde{r_1}, \tilde{x}$	5	$9 \cdot 74$	$8 \cdot 74$

to prove the ownership of a CL signature over m_0, m_t which is always performed.

Our conclusions can be summarized as:

- When the number of attributes is large it can be possible to rely on prime-encoded proofs for improving the issuing process (Section 7.1.2). Moreover, this also applies to the verification of a considerable amount of attributes.
- The selective disclosure operation can be improved in cases where hiding is needed by relying on prime-encoded attributes.
- In this respect, this also applies to equality proofs and pseudonyms relying on prime-encoded attributes.
- By externally and internally reordering the commitments involved in chained AND, OR and NOT operators it can be possible to obtain speed ups of 170.10-644.60 ms.
- By relying on the PRNG in combination of variable reconstruction in RAM we were able to compute multi/-credential proofs within 2 seconds in comparison to the 4 seconds record in literature for the selective disclosure of credentials with one attribute. The same approach was followed for implementing other complex proofs (Sections 6 and 7).

These conclusions can be utilized as guidance in the creation of presentation policies when utilizing contemporary smart cards, taking into account that these operations are computational optimal in the target device in comparison to other implementation options. Finally, other operations used by typical parties in ABC scenarios such as a revocation agent or the inspector can be implemented using the strategies that we have described in this manuscript [8]. Furthermore, complex proofs such as verifiable encryptions and range proofs could be adapted to smart cards relying on these techniques [6, 15].

10 Acknowledgments

I would like to thank Pim Vullers for his advice on programming MULTOS cards and to Gergely Alpár, who participated in the first discussions of this work.

References

1. Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering*, 3(2):111–128, 2013.
2. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *ACM Conference on Computer and Communications Security*, pages 1087–1098, 2013.
3. E. Barker and J. Kelsey. NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators, 2012.
4. Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous credentials on a standard Java Card. In *ACM Conference on Computer and Communications Security*, pages 600–610, 2009.
5. Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.
6. Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2008.
7. Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. Concepts and languages for privacy-preserving attribute-based authentication. *J. Inf. Sec. Appl.*, 19(1):25–44, 2014.
8. Jan Camenisch, Maria Dubovitskaya, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. Concepts and languages for privacy-preserving attribute-based authentication. In *IDMAN*, pages 34–52, 2013.
9. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials (extended version). *IACR Cryptology ePrint Archive*, 2010:496, 2010.

10. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. *ACM Trans. Inf. Syst. Secur.*, 15(1):4, 2012.
11. Jan Camenisch and Els Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *ACM Conference on Computer and Communications Security*, pages 21–30, 2002.
12. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, pages 93–118, London, UK, UK, 2001. Springer-Verlag.
13. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Proceedings of the 3rd international conference on Security in communication networks*, SCN'02, pages 268–289, Berlin, Heidelberg, 2003. Springer-Verlag.
14. Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO*, pages 413–430, 1999.
15. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.
16. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, pages 410–424, 1997.
17. David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
18. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
19. Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 418–430. Springer Berlin Heidelberg, 2000.
20. Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.
21. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
22. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30, 1997.
23. Gesine Hinterwälder, Felix Riek, and Christof Paar. Efficient e-cash with attributes on MULTOS smartcards. In *Radio Frequency Identification. Security and Privacy Issues - 11th International Workshop, RFIDsec 2015, New York, NY, USA, June 23-24, 2015, Revised Selected Papers*, pages 141–155, 2015.
24. R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 230–235, Washington, DC, USA, 1989. IEEE Computer Society.
25. Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.
26. M Luby and C Rackoff. Pseudo-random permutation generators and cryptographic composition. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 356–363, New York, NY, USA, 1986. ACM.
27. Wojciech Mostowski and Pim Vullers. Efficient U-Prove implementation for anonymous credentials on smart cards. In Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis, editors, *SecureComm*, volume 96 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 243–260. Springer, 2011.
28. Michael Sterckx, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Efficient implementation of anonymous credentials on java card smart cards. In *1st IEEE International Workshop on Information Forensics and Security (WIFS 2009)*, pages 106–110, London,UK, 2009. IEEE.
29. Pim Vullers and Gergely Alpár. Efficient selective disclosure on smart cards using Idemix. In Simone Fischer-H editor, *3rd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, IDMAN 2013, London, UK, April 8-9, 2013. Proceedings*.