

# Towards a Full-Featured Implementation of Attribute Based Credentials on Smart Cards<sup>\*</sup>

Antonio de la Piedra, Jaap-Henk Hoepman, Pim Vullers

Radboud University Nijmegen, ICIS DS, Nijmegen, The Netherlands  
{a.delapiedra, jhh, pim}@cs.ru.nl

**Abstract.** Attribute-based Credentials (ABCs) allow citizens to prove certain properties about themselves without necessarily revealing their full identity. Smart cards are an attractive container for such credentials, for security and privacy reasons. But their limited processing power and random access storage capacity pose a severe challenge. Recently, we, the IRMA team, managed to fully implement a limited subset of the Idemix ABC system on a smart card, with acceptable running times. In this paper we extend this functionality by overcoming the main hurdle: limited RAM. We implement an efficient extended Pseudo-Random Number Generator (PRNG) for recomputing pseudorandomness and reconstructing variables. Using this we implement Idemix standard and domain pseudonyms, AND proofs based on prime-encoded attributes, and equality proofs of representation modulo a composite, together with terminal verification and secure messaging. In contrast to prior work that only addressed the verification of one credential with only one attribute (particularly, the master secret), we can now perform multi-credential proofs on credentials of 5 attributes and complex proofs in reasonable time. We provide a detailed performance analysis and compare our results to other approaches.

## 1 Introduction

In Europe several eID systems supporting electronic transactions (particularly in Austria, Belgium, Estonia, Finland, Germany, Netherlands, Portugal, Spain and Sweden) exist. These systems vary a lot in terms of functionality, security levels, and the amount of privacy protection and user control they offer [28,32], and are extensively studied in projects such as Future ID and STORK [1,2]. Using eIDs, the massive utilization of username/password pairs on the Internet could be replaced by a Single-Sign-On (SSO) approach. For this reason a wide range of identity federation systems have been proposed in the last decade, such as SAML [20] and OpenID [33]. These systems rely on a three-party scheme, distinguishing a user, a Service Provider (SP), and an Identity Provider (IdP). To access the service, the user logs in to the IdP, who provides an authentication token to the SP containing all the necessary information. Several security and privacy risks associated with this approach have been identified [30]. For example, the IdP knows every transaction its user performs [5].

Attribute Based Credentials (ABC) solve these problems, as follows. Credentials are secure containers of attributes, that can be selectively disclosed to the SP by the user. Moreover, the underlying cryptographic protocols are unlinkable, ensuring that repeated use of the same credential cannot be linked to one another. Despite the growing number of ABC systems (e.g. [6,10,14]),

---

<sup>\*</sup> The work described in this paper has been supported under the ICT theme of the Cooperation Programme of the 7th Framework Programme of the European Commission, GA number 318424 (Future ID). This research is conducted within the Privacy and Identity Lab (PI.lab) and funded by SIDN.nl (<http://www.sidn.nl>). The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

there are very few applications of such systems in practice. ABC systems rely on the secrecy of a master secret that binds all credentials to a single user. For security and privacy reasons it is therefore preferable to store this master secret and the associated credentials on a smart card, and run all cryptographic protocols (e.g. for issuing credentials and selectively disclosing attributes) on the smart card itself. This way the master secret never leaves the card. However, the limited processing power and random access storage capacity of the smart card pose a severe challenge for implementing the complex cryptographic protocols. Only recently, the IRMA<sup>1</sup> team, managed to fully implement a limited subset of the Idemix ABC system on a smart card, with acceptable running times [36]. In this paper we extend this functionality by overcoming the main hurdle: limited RAM.

**Our contributions.** We first describe, in Section 2, different architectures and implementations of attribute based credentials presented in the literature. Then, in Section 3 we introduce the actual capabilities of the IRMA card, its execution model and its current limitations. In Section 4, we present the design of a PRNG for reducing the RAM requirements of the execution of proofs of knowledge in smart cards. This enables us to extend the number of attributes per credential, beyond the current IRMA limit of 5 attributes per credential. Our construction for recomputing randomness incurs an overhead of only 39.81 ms (3.66 %). Next, we broaden that approach to implement standard and domain pseudonyms in Section 5. Standard pseudonyms add 401.60 ms overhead, while domain pseudonyms in combination with standard pseudonyms add 658.63 ms overhead. Moreover, we can now perform equality proofs of representation across different credentials by combining this technique with variable reconstruction in RAM (e.g. 2,261.19 ms in the best case across 2 credentials, while the rest of proposed works in the literature only addressed the case of verifying one credential with one attribute (namely, the master secret) and obtained performance figures beyond 4 and 7 seconds [9, 35]). This is discussed in Section 6. We implement the terminal verification and a secure channel for ABCs from [4] in Section 7. For each of these constructions we provide a detailed performance analysis, providing performance figures of our own constructions as well as giving those of alternative approaches (tested on the same hardware), and comparing the results. Finally, we provide a performance analysis of all these operations using prime-encoded attributes [13] in Appendix A. In this respect, Bichsel et al. estimated an extra computation time of 1,684 ms over their verification operation with one attribute (i.e. the master secret) using a modulus of 1,536 bits in the prime-encoded AND proof [9]. That is  $1.684 + 10.550 = 12.234$  seconds. In our case (Appendix A) we can perform AND proofs with credentials of 5 attributes in 1,214.24 ms (hiding all the attributes) and we only require 1,547.83 ms for proving the ownership of the 5 attributes revealing them all using a modulus of 1,024 bits. In this case, we required 1,579.53 ms when proving the ownership of a pseudonym. Moreover, we show that the computation of equality proofs of representation involving several credentials on the smart card is also feasible relying on prime-encoding attributes.

## 2 Related work

Generally, private ABCs rely on the idea of a blind and randomizable signature over a set of attributes [21]. A user that owns this type of signature can perform authentication operations by selectively disclosing a subset of attributes that describes her digital identity. Moreover, it is not possible to link transactions to signatures. Modern anonymous credential systems such as Idemix [14] and U-Prove [10] rely on that building block in combination with proofs of knowledge [22]. In the last few years, a myriad of credential systems has been implemented in smart cards [8, 9, 31, 35, 36]. Due to their availability in the market and their functionalities, the Java

<sup>1</sup> <https://www.irmacard.org> (accessed August 11, 2014)

Card and MULTOS platforms are generally the preferred targets. Bichsel et al. presented the first implementation of Idemix on the Java Card platform in 2009. Proving the possession of one credential with one attribute (i.e. the master secret), required 7.4 seconds (1,280-bit modulus) and 10.55 seconds with a modulus of 1,536 bits [9]. Sterckx et al. followed a similar approach for implementing the signing protocol of Direct Anonymous Attestation (DAA) in Java Cards [35]. In their design, one transaction requires 4.2 seconds using a modulus of 1,024 bits. Nonetheless, the Java Cards 2 that Bichsel et al. and Sterckx et al. relied on do not provide direct access to modular arithmetic operations. Consequently, these authors had to rely on different strategies for performing modular multiplications and exponentiations, thus undermining the overall performance of the implementation. The IRMA card is based on the MULTOS platform embedded on the Infineon SLE78 chip. In contrast, this chip supports a variety of modular arithmetic operations that are crucial in the implementation of anonymous credentials together with asymmetric encryption primitives (RSA), signature schemes (RSA, ECDSA), symmetric encryption techniques (AES, 3DES) and hashing algorithms (SHA-1, SHA-2). Vullers et al. implemented in this platform the issuing and selective disclosure operations of Idemix. Using credentials of 5 attributes, the disclosure of all the attributes requires 0.947 seconds whereas the worst case (hiding the 5 attributes) is performed in 1.454 seconds [36].

### 3 The IRMA card

IRMA relies on the specification of the Identity Mixer Anonymous Credential System [14] and is the first full card (i.e. no off-card or precomputation based) implementation of Idemix on smart cards suitable for real life transactions i.e the performance of a typical operation is reduced to 1–1.5 seconds [36] in comparison to the first attempts for implementing anonymous credentials in the literature (Section 2). Idemix provides different functionalities for proving the possession of attribute-based credentials and their properties e.g. [12, 18].

Idemix relies on different cryptographic blocks such as the Camenisch-Lysyanskaya (CL) signature, secure under the Strong RSA assumption [16]. 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.

After the issuing process, the users owns a CL signature over one credential, represented by the triple  $(A, e, v)$  over  $(m_0, m_1, \dots, m_5)$ . This information is stored in the card for each credential. The CL signature is created by an issuer according to its public key  $(S, Z, R_0, R_1, \dots, R_5 \in QR_n, n)$  using its secret key  $(p, q)$ . For instance, a CL signature over a set of attributes  $(m_0, \dots, m_5)$  is computed by selecting  $A, e$  and  $v$  s.t.  $A^e = ZR_0^{-m_0}R_1^{-m_1}R_2^{-m_2}R_3^{-m_3}R_4^{-m_4}R_5^{-m_5}S^{-v} \pmod n$ . Then, a third party can check the validity of the signature by using the issuer's public key and the tuple  $(A, e, v)$  as  $Z \equiv A^e R_0^{m_0} R_1^{m_1} R_2^{m_2} R_3^{m_3} R_4^{m_4} R_5^{m_5} S^v \pmod n$ . In IRMA, for performance reasons, the size of the modulus  $n$  is restricted to  $l_n = 1,024$  bits whereas the attributes are represented as  $l_m = 256$  bits. The rest of parameters are set as  $l'_e = 120$ ,  $l_\phi = 80$ ,  $l_H = 256$ ,  $l_e = 504$ , and  $l_v = 1,604$  bits<sup>2</sup>.

The key property of the CL signature in Idemix is to prove its possession without revealing additional information and performing the selective disclosure of the cardholder's attributes (Protocol 1) via discrete logarithm representation modulo a composite proofs of knowledge [25].

<sup>2</sup> The term  $l'_e$  represents the size of the interval where the  $e$  values are selected,  $l_\phi$  is the security parameter of the statistical ZKP, 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.

Protocol 1: Message flow for proving the ownership of a CL signature over a set of attributes

Prover	Public	Verifier
$Z \prod_{i \in A_r} R_i^{-m_i} = A'^e S^{v'} \prod_{i \in A_{\bar{r}}} R_i^{m_i}$	$S, Z, R_0, R_1, \dots, R_5 \in QR_n, n$	$A', v', m_{i \in A_r}$
<p><b>Signature randomization</b></p> $r_A \in_R \{0, 1\}^{l_n + l_o}$ $A' = AS^{r_A} \pmod n$ $v' = v - er_A$ $e' = e - 2^{l_e - 1}$ <p><b>Generation of <math>t</math>-values</b></p> $\tilde{e} \in_R \pm \{0, 1\}^{l_e + l_o + l_H}$ $\tilde{v}' \in_R \pm \{0, 1\}^{l_v + l_o + l_H}$ $\tilde{m}_i \in_R \pm \{0, 1\}^{l_m + l_o + l_H} (i \in A_{\bar{r}})$ $\tilde{Z} = A'^{\tilde{e}} (\prod_{i \in A_{\bar{r}}} R_i^{\tilde{m}_i}) S^{\tilde{v}'}$ <p><b>Generation of <math>s</math>-values</b></p> $\hat{e} = \tilde{e} + ce'$ $\hat{v}' = \tilde{v}' + cv'$ $\hat{m}_i = \tilde{m}_i + cm_i (i \in A_{\bar{r}})$	$\xrightarrow{\tilde{Z}}$ $\xleftarrow{c}$  $\xrightarrow{\hat{e}, \hat{v}', \{\hat{m}_i\}_{i \in A_{\bar{r}}}}$	$c \in_R \{0, 1\}^{l_H}$  $A'^{\hat{e}} S^{\hat{v}'} \prod_{i \in A_{\bar{r}}} R_i^{\hat{m}_i} \stackrel{?}{=} \tilde{Z} (Z \prod_{i \in A_r} R_i^{-m_i})^c$

The typical 3-movement protocol (commitment, challenge and response) depicted in Protocol 1 is transformed into a Non-Interactive Proof of Knowledge (NIZK) via the Fiat-Shamir heuristic [24]. Therefore, the challenge  $c$  is computed by the card via a collision-resistant hash-function over the commitments and common values. Accordingly, an empty proof of possession over a set of attributes  $(m_0, \dots, m_5)$  is represented using the Camenisch-Staedler notation [19] as: NIZK:  $\{(\varepsilon', \nu', \alpha_0, \dots, \alpha_5) : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \pmod n\}$  being the Greek letters  $(\varepsilon', \nu')$  and  $(\alpha_0, \dots, \alpha_5)$  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:  $\{(\varepsilon', \nu', \alpha_0, \alpha_2, \alpha_3, \alpha_4, \alpha_5) : Z R_1^{-m_1} \equiv \pm R_0^{\alpha_0} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \pmod n\}$ .

In IRMA, the prover part of Idemix is implemented in the card as a set of states (PROVE\_CREDENTIAL, PROVE\_COMMITMENT, PROVE\_SIGNATURE and PROVE\_ATTRIBUTE) that mimics the Prover-Verifier interaction between a terminal and the smart card<sup>3</sup>. In each transaction, both entities exchange ISO 7816 APDUs that retrieve and write data in the smart card volatile (RAM) and non-volatile (EEPROM) memories [27]. When the card receives a verification request, it changes its initial state to PROVE\_CREDENTIAL. Then, it acquires a presentation policy with the description of the attributes that must be revealed (i.e. those  $\{m_i\}_{i \in A_r}$ ) and hidden (i.e.  $\{m_i\}_{i \in A_{\bar{r}}}$ ). Then, the card performs the operations depicted in Protocol 1 (PROVE\_COMMITMENT). Afterwards, the card changes its working state to PROVE\_SIGNATURE. In this state, the verifier can request the randomized tuple  $(A', \hat{e}, \hat{v}')$ . Finally, the card switches to PROVE\_ATTRIBUTE, where the verifier is allowed to request the set of revealed and hidden attributes related to the proof.

<sup>3</sup> We refer the reader to [36] for a description about how a  $(A, e, v)$  triple is obtained by the card.

### 3.1 Execution model

The latency of the verification operation can be modeled first according to the number of attributes per credential ( $n = 5$  in the case of IRMA) together with the number of attributes that are revealed ( $r$ ) or hidden. If we consider the worst case (all the attributes are hidden),  $n - r + 1$  extra computations will be required for generating each  $\tilde{m}_i$  value (one extra operation is considered since the master secret is always hidden). Otherwise, the  $m_i$  attributes are sent in clear to the verifier. Eq. 1 represents the overall latency of the four states described above according to the  $(n, r)$  parameters.

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

The time that  $T_{sel\_cred}$  comprises is related to the PROVE\_CREDENTIAL state whereas  $T_{gen\_commit}(n, r)$  represents PROVE\_COMMITMENT and  $T_{get\_sig}(i) \setminus T_{get\_attr}(i)$  are related to the PROVE\_SIGNATURE and PROVE\_ATTRIBUTE states respectively. Furthermore, the latency of the PROVE\_COMMITMENT state can be expanded to the following expression:

$$T_{gen\_commit}(n, r) = \sum_{i=A,v} T_{rand\_sig}(i) + T_{gen\_t\_values}(n-r+1) + T_{hash} + T_{gen\_s\_values}(n-r+1) \quad (2)$$

Eq. 2 represents the randomization of the CL signature and the generation of the  $t\_values$ ,  $s\_values$  and the challenge  $c$ . Further,  $T_{gen\_t\_values}$  represents the latency due to the computation of the commitment according to the number of non-disclosed values i.e.  $\sum_{i=1}^{n-r+1} T_{mul\_exp}(R_i^{\tilde{m}_i})$ . This value is then multiplied by  $A^{\tilde{e}} \cdot S^{\tilde{v}'}$  as described in Section 3. In addition,  $n - r$  random  $\tilde{m}_i$  values must be generated. Finally, the  $s\_values$  are generated according to the number of hidden attributes:

$$T_{gen\_s\_values}(n - r + 1) = T_{gen\_e} + T_{gen\_v'} + \sum_{i=0}^r T_{gen\_m_i} \quad (3)$$

From the Equations 1-3 we notice the following. First, that 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  $T_{gen\_s\_values}$ . Second, that the overall verification time 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 random 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.

### 3.2 Memory model

In the current IRMA implementation, the CL signatures and the credential attributes are stored in EEPROM. Besides, the intermediate values  $\tilde{m}_i$  and the randomized signature are computed in RAM in order to speed-up the overall performance of the verification operation. Since the RAM only comprises 960 bytes (together with 1,160 bytes of transient memory), we must consider how to scale the operations with credentials of a large number of attributes in order to deal with those intermediate values. In this respect, independently storing each  $\hat{m}_i$  value for large credentials is impossible due to the RAM restrictions. In the current implementation, based on  $5 + 1 \hat{m}_i$  values (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. Finally, given the current memory utilization, any optimization should be based on rearranging the storage of the random  $\tilde{m}_i$  values.

## 4 Preliminary optimizations

Our goal is to generate as many  $\hat{m}_i$  values as needed without being limited by the current RAM size. This would make it possible to operate with larger credentials in the card. As noted before, these values are used in two parts of the generation of the NIZK: (1) during the computation of the commitment and (2) for hiding the desired attributes in the generation of the  $s\_values$ . In the seminal paper of Bichsel et al. they 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) [9]. Consequently, we can extend this approach to regenerate all the involved pseudorandomness, not only the random exponents, for supporting credentials with a large number of attributes and implementing: pseudonyms, domain pseudonyms and AND proofs. Moreover, we coupled this technique with variable reconstruction in RAM for making possible to compute multi-credential proofs in the case of the equality proofs of representation.

Using cryptographic primitives such as block ciphers and Message Authentication Codes (MACs), conjectured as pseudorandom generators under the assumption that one-way functions exist (cf. [26, 29]), a wide range of PRNGs has been proposed in the literature e.g. [7, 23]. Therefore, it is expected that the output of these constructions would be indistinguishable from random by any probabilistic polynomial time algorithm or distinguisher. Among the schemes described in [7, 23] (e.g. Fortuna, HMAC\_DBRG and HASH\_DBRG), all share the same behaviour: (1) acquire new entropy, (2) process the entropy into a seed and, if needed, add a personalizing string, and (3) generate pseudorandom bits using a cryptographic primitive (e.g. a block cipher) categorized as Generator Function (GF). The HASH\_DBRG PRNG utilizes SHA-1/-2 for generating pseudorandomness whereas the HMAC\_DBRG PRNG can optionally rely on one of those primitives following the HMAC construction, where a key  $k$  is also part of the initial state. Finally, a PRNG can be constructed using a block cipher such as Fortuna [23], which only enciphers a counter using a key derived from an entropy input and processed via SHA-256.

In Table 1, we present the performance results in our target device for computing one  $\hat{m}_i$  value (74 bytes) using six different PRNGs. We also present the number of calls to the GF in each case to generate  $|\hat{m}_i|$  bytes. Notice that the PRNGs based on HMACs are considerably slower due to the fact that they require performing two hash operations per call together with two updating functions [7]. Moreover, the performance of using Fortuna and the PRNG based on SHA-1 is similar, given that the number of calls to the SHA-1 hash algorithm is almost equal in both cases (26.33 ms and 29.32 ms respectively). However, when a considerable amount of pseudorandomness per session is generated (particularly, during the execution of a proof that involves more than one credential), a difference of 3 ms can be significant<sup>4</sup>. In this respect, we have lowered the security level to AES-128 (24.76 ms per  $\hat{m}_i$ ).

In our case, during each verification 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, this seed is fed into the GF (AES-128) as the key. At the beginning of each verification session 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. Thus, this design provides backtracking resistance between verification sessions. Moreover, prediction resistance is ensured if we rely on the security of the

<sup>4</sup> See, for instance, the number of required calls in our approach for performing equality proofs of representation (Section 6, Table 4). In that case, only generating 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 if all the attributes are hidden.

Table 1: 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)
[7]	HMAC_DBRG	SHA-1	20	12	48.64
[7]	HASH_DBRG	SHA-1	20	5	26.33
[7]	HMAC_DBRG	SHA-256	32	10	106.78
[7]	HASH_DBRG	SHA-256	32	4	47.47
[23]	Fortuna	AES-256	16	5	29.32
This work	IRMA	AES-128	16	5	24.76

AES block cipher. The PRNG runs the following sequence in this case:  $init_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow reset_{PRNG}() \Rightarrow \tilde{m}_i$ .

#### 4.1 Results for verifying a full credential

In our implementation we proceed as follows. We replace the former  $74 \cdot 6$  bytes for storing all the  $\hat{m}_i$  values by three values maintained in RAM during the verification session: 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 process is repeated two times. First, during the computation of the  $t\_value$  and second, during the  $s\_values$  generation. In the second part, the PRNG is reset to its initial state (by choosing  $c = 0$  again) in order to obtain the same output as the first time without storing all the reconstructed  $\hat{m}$  values. 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. This is depicted in Table 2<sup>5</sup>. We have considered both worst cases (WC, where all the attributes are hidden) and best cases (BC, where only the master secret ( $m_0$ ) is hidden).

Table 2: Performance overhead while verifying five attributes using a custom PRNG for generating the  $\hat{m}$  values (ms)

Work	Encoding	Case	$T_{sel\_cred}$	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(m_0, \dots, m_5)$	Total
This work	normal	BC	103.10	840.73	14.12, 11.52, 19.54	38.32, 11.51*5	1,084.91
This work	normal	WC	104.95	1,307.35	15.11, 11.48, 19.49	38.30*6	1,688.20
[36]	normal	BC	104.12	826.25	14.16, 11.48, 19.50	12.10, 11.49*5	1,045.10
[36]	normal	WC	105.20	1,259.24	16.10, 11.49, 19.48	12.13*6	1,484.32

According to Eq. 1–3, the calculation of the  $\hat{m}$  values as  $\tilde{m}_i + cm_i$  was performed in  $T_{gen\_commit}(n, r)$ . Now, that generation operation is performed on demand in  $T_{get\_attr}(i)$  i.e.

<sup>5</sup> The notation and abbreviations in Tables 2–3 and 5–9 are utilized as follows. First, for those latencies that are related to the same operation e.g.  $T_{get\_attr}(i)$ , only one element appears multiplied by the number of elements of its type that are involved e.g.  $11.51 * 5$ . Then, for each element that is optional according to the different proofs represented in the same table, a vertical bar (|) appears in the header of the table (e.g. Table 3). For those values that do not belong to the proof an horizontal bar (–) is utilized. Finally, we represent the best performance figures in **bold**.

when the verifier asks for those values. Therefore,  $T_{gen\_s\_values}$  is represented as  $T_{gen\_s\_values} = T_{gen\_ê} + T_{gen\_ô'}$  and  $\sum_{i=1}^n T_{get\_attr}(i)$  includes the generation of  $\hat{m}_i$  as  $T_{gen\_{\hat{m}_i}}$  for  $r$   $\hat{m}_i$  values. This means, that we reduce the overall time in  $T_{gen\_commit}(n, r)$  but add an extra delay according to the random generation of the  $\tilde{m}_i$  values. We also obtained an extra delay in  $T_{get\_attr}(i)$ , consisting of (1) recomputing the pseudorandomness for each  $\tilde{m}_i$  value and (2) computing  $\hat{m}_i = \tilde{m}_i + cm_i$ . All in all, we obtained a reduction of RAM of 338 bytes with an added overall latency of 203.88 ms in the worst case whereas the extra latency in the best case is only restricted to the generation of the  $\hat{m}_i$  value for hiding the master secret, where only 39.81 ms are required<sup>6</sup>.

## 5 Implementation of standard and domain pseudonyms

Given the optimizations described in the past section, now it is possible to implement additional proofs in combination to proving the ownership of a CL signature over a set of attributes. In this section, we relate our implementation of Idemix pseudonyms [15]. They provide extended operations to basic protocols such as issuing a credential associated to a pseudonym or connect the cardholder's verification process to a given pseudonym. The latter, would guarantee being recognized the next time an user visited the same SP. Besides standard pseudonyms (described as randomized commitments to the cardholder's master secret i.e.  $\text{Nym} = g^{m_0} h^r \pmod{\Gamma}$  where both generators  $(g, h)$  and modulo  $\Gamma$  are public and part of the system group parameters) it is possible to create pseudonyms associated to a certain domain such as an organization. They are derived as  $\text{dNym} = g_{dom}^{m_0}$  where  $g_{dom} = \mathcal{H}(\text{dom})^{(\Gamma-1)/\rho}$  and the group  $Z_{\Gamma}^*$  has order  $\Gamma - 1 = \rho \cdot b$  for a prime  $\rho$  [15]. For instance, proving the ownership of both a standard and domain pseudonyms can be performed in zero knowledge as NIZK:  $\{(\varepsilon', \nu', \alpha_0, \dots, \alpha_5, \psi) : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} R_2^{\alpha_2} R_3^{\alpha_3} R_4^{\alpha_4} R_5^{\alpha_5} A^{\varepsilon'} S^{\nu'} \pmod{n} \wedge \text{nym} \equiv g^{\alpha_0} h^{\psi} \pmod{\Gamma} \wedge \text{dNym} \equiv g_{dom}^{\alpha_0} \pmod{\Gamma}\}$  without revealing any attribute. However, performing a certain degree of selective disclosure would provide the SP with more identification details linked to the pseudonym. In order to design a RAM-efficient implementation of standard/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.

### 5.1 Results for verifying a full credential with an associated pseudonym

The user must store in EEPROM the two generators  $(g, h)$ , together with  $r$  and the modulus  $\Gamma$ . Given that our target device is comprised of 80KB of EEPROM and the current implementation is 31,897 bytes we have plenty of space for storing different pseudonyms. In this respect, encoding pseudonyms as strings of 32 bytes would allow to store up to  $80\text{K} - (31,897/32) = 1,503$  pseudonyms in the card. Finally, in addition to the revealed or hidden attributes that the cardholder sends to the verifier, the commitments  $\text{nym}$  and  $\text{dNym}$  together with the  $s\_value$   $\hat{r}$  are recomputed and their respective delay is added to  $T_{get\_attr}(i)$ . As in Table 2, we have represented the best case/worst case scenarios for each type of pseudonym in Table 3. In comparison to Table 2, performing the extra number of modular exponentiations related to the pseudonyms commitments (e.g.  $\text{nym}$ ) required  $1,176.02 - 840.73 = 335.29$  ms in the best case. However, due to the optimizations described in Section 4, is also possible to store extra commitments in RAM in order to avoid recomputing them during  $T_{get\_attr}$ .

<sup>6</sup> Extending the number of attributes of the credential also involves modifying the issuing protocol implementation. Given the space limits, we opted for showing how to compute complex proofs.

Table 3: Performance analysis of proving the ownership of standard and domain pseudonyms (normal encoding, ms)

Implementation	Case	$T_{sel\_cred}$	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(m_0, \dots, m_5, nym, \hat{r} dNym)$	Total
Nym	BC	103.10	1,176.02	15.30, 11.41, 19.57	38.42, 11.55*5, 13.95, 50.99	<b>1,486.51</b>
Nym	WC	103.15	1,647.33	15.35, 11.54, 19.57	38.11*6, 13.99, 50.86	2,065.42
Nym $\wedge$ dNym	BC	103.01	1,373.37	15.28, 11.41, 19.35	38.39, 11.83*5, 14.02, 58.37   51.12	<b>1,743.54</b>
Nym $\wedge$ dNym	WC	104.23	1,836.00	15.19, 11.59, 19.24	38.02*6, 13.84, 58.32   51.42	2,338.01

## 6 Tailored execution of equality proofs of representation

Sometimes, it is useful to prove that two or more credentials share some values [17]. This type of proof would enable verifiers to evaluate different properties in the credentials of the cardholder, for instance, proving that two or more credentials belong to the same cardholder via the master secret that is included in all the credentials with independence of the issuer. This is essential to prevent credential pooling attacks.

In this section, we restrict ourselves to equality proofs of two credentials, where the ownership of the cardholder is proved through the equality of the master secret, and where independent selective disclosures can be performed in each credential. Given two credentials issued by different issuers over the same master secret ( $m_0$ ) and two different CL signatures ( $A_1, e_1, v_1$ ) and ( $A_2, e_2, v_2$ ), we describe an empty (i.e. where all the attributes of the first and second credentials are hidden) equality proof of this type as: NIZK:  $\{(\varepsilon'_1, \nu'_1, \varepsilon'_2, \nu'_2, \mu, (\alpha_1, \dots, \alpha_5), (\beta_1, \dots, \beta_5)) : Z^{(1)} \equiv \pm R_0^{(1)\mu} R_1^{(1)\alpha_1} R_2^{(1)\alpha_2} R_3^{(1)\alpha_3} R_4^{(1)\alpha_4} R_5^{(1)\alpha_5} A^{(1)\varepsilon'_1} S^{(1)\nu'_1} \pmod{n_1} \wedge Z^{(2)} \equiv \pm R_0^{(2)\mu} R_1^{(2)\beta_1} R_2^{(2)\beta_2} R_3^{(2)\beta_3} R_4^{(2)\beta_4} R_5^{(2)\beta_5} A^{(2)\varepsilon'_2} S^{(2)\nu'_2} \pmod{n_2}\}$ . Where  $\mu$  represents the non-disclosed master secret<sup>7</sup> and the two public keys of the issuers consists of  $(S^{(1)}, Z^{(1)}, R_0^{(1)}, R_1^{(1)}, \dots, R_5^{(1)} \in QR_{n_1}, n_1)$  and  $(S^{(2)}, Z^{(2)}, R_0^{(2)}, R_1^{(2)}, \dots, R_5^{(2)} \in QR_{n_2}, n_2)$ . Moreover, the following  $s\_values$  for the groups of attributes of each credential and the master secret need to be computed:  $\hat{m}_0 = \tilde{m}_0 + cm_0, \hat{m}_1^{(1)} = \tilde{m}_1^{(1)} + cm_1^{(1)}, \hat{m}_2^{(1)} = \tilde{m}_2^{(1)} + cm_2^{(1)}, \hat{m}_3^{(1)} = \tilde{m}_3^{(1)} + cm_3^{(1)}, \hat{m}_4^{(1)} = \tilde{m}_4^{(1)} + cm_4^{(1)}, \hat{m}_5^{(1)} = \tilde{m}_5^{(1)} + cm_5^{(1)}, \hat{m}_1^{(2)} = \tilde{m}_1^{(2)} + cm_1^{(2)}, \hat{m}_2^{(2)} = \tilde{m}_2^{(2)} + cm_2^{(2)}, \hat{m}_3^{(2)} = \tilde{m}_3^{(2)} + cm_3^{(2)}, \hat{m}_4^{(2)} = \tilde{m}_4^{(2)} + cm_4^{(2)}, \hat{m}_5^{(2)} = \tilde{m}_5^{(2)} + cm_5^{(2)}$ .

Finally,  $\hat{v}_1 = \tilde{v}_1 + cv'_1, \hat{v}_2 = \tilde{v}_2 + cv'_2, \hat{e}'_1 = \tilde{e}_1 + ce_1$  and  $\hat{e}'_2 = \tilde{e}_2 + ce_2$  are computed for each CL signature in  $T_{get\_sig}(i)$ .

### 6.1 Design

In order to implement these proofs, we must address three types of requirements in terms of: (1) space, (2) performance and (3) cryptographic capabilities of the card. First, we need space to store and/or maintain in RAM two or more  $(A, e, v)$  tuples in order to generate each  $t\_value$  of the proof. Moreover, we also require space for storing the  $(\hat{e}, \hat{v}')$  tuples for each credential during the computation of each  $s\_value$ . Furthermore, we need to perform all these computations in a reasonable time. In this respect, performing the operations in RAM would be a top priority. Finally, we need a hash primitive for computing multiple and subsequent blocks of data ( $t\_values$ ) in order to generate the challenge  $c$ . In this case, we need to include the set of the  $t\_$  and common values for each credential in the proof. Since 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

<sup>7</sup> In this case  $\alpha_0 = \beta_0$  if both credentials belong to the same cardholder. We represent the non-disclosed master secret as  $\mu$  following the Camenisch-Staedler notation [19].

must find an alternative function that can compute hashes with partial inputs in a subsequent manner.

In order to implement the equality proof on the card and be able to cope with multiple signatures of different issuers we extend the PRNG described in Section 4 and couple it with variable reconstruction in RAM. We notice that the  $(\hat{e}, \hat{v}')$  values only depend on the  $(\tilde{e}, \tilde{v})$  pseudorandom variables. Since they do not depend on  $\tilde{m}$ , the same space reserved in RAM for such value (74 bytes as described in Section 4) 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 the deterministic PRNG  $\hat{e}$  and  $\hat{v}'$  (i.e. as  $\hat{e} = \tilde{e} + ce'$  and  $\hat{v}' = \tilde{v} + cv'$ ) for each credential during the generation of the  $t$ - and  $s$ -values. Furthermore, the randomized computation of the signature component  $A'$  requires  $r_A$ , another random value that can be derived from the PRNG. Moreover, since the randomization of this value is independent from the rest of the signature  $(e, v)$  and the  $\hat{m}_i$  values, we can compute all these variables in a sequentially way. 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 verifier.

Therefore, the generation and recomputing of these values for an equality proof of two credentials would be orchestrated by the PRNG as  $init_{PRNG}() \Rightarrow r_A^{(1)} \Rightarrow \tilde{v}^{(1)} \Rightarrow \tilde{e}^{(1)} \Rightarrow \tilde{m}_i^{(1)} \Rightarrow r_A^{(2)} \Rightarrow \tilde{v}^{(2)} \Rightarrow \tilde{e}^{(2)} \Rightarrow \tilde{m}_i^{(2)} \Rightarrow reset_{PRNG}() \Rightarrow r_A^{(1)} \Rightarrow \tilde{v}^{(1)} \Rightarrow \tilde{e}^{(1)} \Rightarrow \tilde{m}_i^{(1)} \Rightarrow r_A^{(2)} \Rightarrow \tilde{v}^{(2)} \Rightarrow \tilde{e}^{(2)} \Rightarrow \tilde{m}_i^{(2)}$ . We describe two<sup>8</sup> different alternatives for performing this proof according to different scenarios and speed requirements.

**Alternative a: equality proofs across  $n$  credentials** We have depicted in Table 4 the performance of recomputing the randomness for the  $(A', \hat{v}', \hat{e}, \hat{m}_i)$  values and reconstructing their values in RAM. Despite the required number of calls to the PRNG is higher in  $\hat{v}'$ , the overall execution time is dominated by the reconstruction of  $A'$  that 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 4: Time required for reconstructing  $A'_i, \hat{v}'_i, \hat{e}_i,$  and  $\hat{m}_i$  in RAM

Variable	Operation	Size (bytes)	No. of calls to PRNG	Delay (ms)
$A'_i$	$AS^{r_A}$	138	9	235.191
$\hat{v}'_i$	$v' = v - e \cdot r_A$	255	16	104.365
	$\hat{v} = \hat{v}' + c \cdot v'$			
$\hat{e}_i$	$\tilde{e} + c \cdot e'$	57	4	30.710
$\hat{m}_i$	$\tilde{m}_i + c \cdot m_i$	74	5	36.708

Finally, in relation to the third requirement, we rely on the `PRIM_SECURE_HASH_IV` primitive of the MULTOS card in order to subsequently hash each  $t$ -value. This primitive makes possible to avoid maintaining a long string of bytes in RAM with all the required inputs for generating the challenge. Therefore, each  $A'$  and  $t$ -value is generated in an iterative way and sequentially added to the temporary digest. After the last  $t$ -value, 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

<sup>8</sup> We provide a third alternative via prime encoding in Appendix A.

now recomputed on demand when the verifier request them. Consequently, that latency is added to  $T_{get\_sig}(i)$ .

**Alternative b: equality proofs across 2 credentials** In this alternative, we work under the assumption that each card stores only two credentials i.e. one root credential with different information about an issuing organization, an expiration date or a revocation state together with a second credential that includes the cardholders attributes. In both credentials the master secret is shared and an equality proof can be performed across the two in order to proof the validity of the card or the attributes. In this case, it can be possible to store both  $A^{(1)}$  and  $A^{(2)}$  and avoid recomputing them two times as described in the first alternative (Table 4). Moreover, the randomization factors  $r_A^{(1)}$  and  $r_A^{(2)}$  can be stored too in order to avoid regenerate them via the PRNG during the computing of  $\hat{v}'_i$ . In this case, we use the transient memory of the card for storing these four values. Given that its size is 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. In this respect, we need  $2 \cdot 128$  bytes for  $A^{(1)}$ ,  $A^{(2)}$  and  $2 \cdot 138$  bytes for  $r_A^{(1)}$  and  $r_A^{(2)}$  in our case. Finally, 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).

## 6.2 Results for performing an equality proof of representation with two full credentials

We have depicted in Table 5<sup>9</sup> the performance of equality proofs using the two described alternatives (a, b) for proving that the credentials (2 in this example) of the cardholder share their master secret and therefore, are linked to her.

Table 5: Performance overhead while verifying two credentials with 5 attributes using the equality proof (normal encoding, ms)

Alternative	Case	$T_{sel\_cred}$ (ms)	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(m_0, \dots, m_5)$	Total
<b>a</b>	BC	104.12	1,805.24	$(231.95, 91.63, 27.47)^{(1,2)}$	$(32.02, 10.11*5)^{(1)}, (10.10*5)^{(2)}$	2,744.51
<b>a</b>	WC	105.23	2,738.78	$(228.38, 91.64, 27.55)^{(1,2)}$	$(31.50*6)^{(1)}, (31.13*5)^{(2)}$	3,883.83
<b>b</b>	BC	103.99	1,743.37	$(17.82, 91.58, 27.40)^{(1,2)}$	$(32.05, 10.49*5)^{(1)}, (10.53*5)^{(2)}$	<b>2,261.19</b>
<b>b</b>	WC	103.47	2,673.60	$(14.78, 91.25, 27.37)^{(1,2)}$	$(31.86*6)^{(1)}, (31.19*5)^{(2)}$	3,390.60

Using the second alternative (b), 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. Finally, the first alternative (a), due to the fact that we recompute  $(\hat{e}, \hat{v}')$  for each credential in the generation of each  $t\_value$ , increases the execution time of  $T_{gen\_commit}(n, r)$  from 840.73 ms (verification of one credential, best case, Table 2) to 1,805.24 ms (best case, all the attributes are revealed) and from 1,307.35 ms (verification of one credential, all the attributes

<sup>9</sup> We use the superscripts 1 and 2 for referring to the operations related to the credentials 1 and 2 of the equality proof. During the randomization of the CL signatures the operations for each credential are the same. We have put together the operations related to each credential in the worst case. Therefore, two pairs of 5 attributes are hidden together with the master secret i.e.  $6 + 5$  operations if the master secret is hidden during the computation of the  $s\_values$  of the first credential.

hidden, Table 2) to 2,738.78 ms (worst case, all the attributes remain hidden). However, given the case that the user is requested to perform an equality proof of her credentials, it would be rare to hide all the attributes in the case that one of the credentials (e.g. a root credential) would contain information about the issuing operation required to be revealed e.g. a date, the name of an organization, etc.

## 7 Authenticated secure channel

We have depicted in Table 6 the results for performing all the operations described in Sections 4-6 through terminal verification and secure channel. We rely on the secure channel for ABCs proposed by Alpár et al. in [4] and we perform terminal verification via ECDSA signatures using the light secp160r1 (160 bits) curve [34]. Besides, we rely on the normative for secure messaging of the German ID [11] for providing authentication and confidentiality (CBC-MAC and 3DES-CBC are used [11]). If we compare our results depicted in Table 6 with the works described in Section 2, Bichsel et al. required 7.4 seconds for verifying a credential of one attribute (i.e. master secret, modulo 1,280 bits) whereas we can perform an equality proof of 5 credentials in the same time (Appendix A, Figure 1). Besides, one transaction in the implementation of Sterckx et al. required 4.25 s using a modulus of 1,024 bits whereas we can perform all the operations described in Sections 4–6 (best cases) within the same time <sup>10</sup>.

Table 6: Performance analysis of a full operation using terminal verification and secure channel (normal encoding, ms)

Operation	Case	$T_{set\_sc}$	$T_{sel\_cred}$	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v_1, v_2)$	$T_{get\_attr}(m_0, \dots, m_5   nym, \hat{r}   dNym)$	Total
<b>verify 1 cred</b>	BC	203.31	183.50	889.53	82.49, 49.40, 81.25, 68.82	76.04, 49.40*5	<b>1,881.30</b>
<b>verify 1 cred</b>	WC	203.29	183.53	1,360.42	82.47, 49.38, 81.24, 68.80	76.14*6	2,484.10
Nym	BC	203.32	185.26	1,226.26	81.99, 49.21, 80.88, 68.54	75.75, 49.28*5   81.01, 93.62	<b>2,392.21</b>
Nym	WC	203.28	182.27	1,690.50	82.18, 49.35, 81.01, 68.57	75.70*6   81.32, 93.67	2,986.42
Nym $\wedge$ dNym	BC	203.29	182.41	1,419.52	82.19, 49.43, 81.02, 68.63	75.97, 49.41*5   81.38, 93.63   124.23	<b>2,708.82</b>
Nym $\wedge$ dNym	WC	203.31	182.56	1,893.67	82.02, 49.45, 81.02, 68.50	75.08*6   81.21, 93.75   124.26	3,310.23
<b>eq. proof b</b>	BC	203.31	182.56	1,809.39	(82.10, 49.41, 81.05, 68.54) <sup>1,2</sup>	83.37, (49.99*5) <sup>1,2</sup>	<b>3,340.71</b>
<b>eq. proof b</b>	WC	203.33	182.56	2,743.20	(82.12, 49.43, 81.02, 68.50) <sup>1,2</sup>	84.10, (84.33*5) <sup>1,2</sup>	4,618.62

We have made available our prototypes<sup>11</sup> for public verifiability under the General Public License (GPL) together with a terminal code based on the CHARM cryptographic framework [3].

## 8 Conclusions

We have presented the performance evaluation and our design options for implementing Idemix on a smart card together with a variety of operations for executing complex proofs. We relied on recomputing all the involved pseudorandomness using a PRNG. Moreover, we have described our results in combination with a secure channel coupled with terminal verification based on ECC. All our operations required between 1–3.3 seconds (best cases) and between 1–4.6 (all cases) while the prior art only addressed the case of one credential with one attribute (i.e. the master secret). In contrast, our performance figures can be acceptable in on-line settings and could be adapted to off-line scenarios.

<sup>10</sup> We also note that these results can be probably reproduced in other devices relying on the Infineon SLE78 chip.

<sup>11</sup> [https://github.com/adelapie/irma\\_phase\\_2](https://github.com/adelapie/irma_phase_2) (accessed August 11, 2014)

## References

1. EU FP7 Future ID (accessed August 11, 2014). <http://www.futureid.eu/>, 2014.
2. EU FP7 Secure identity across borders linked (STORK) 2.0 (accessed August 11, 2014). <https://www.eid-stork2.eu/>, 2014.
3. 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.
4. Gergely Alpár and Jaap-Henk Hoepman. A secure channel for attribute-based credentials: [short paper]. In *Digital Identity Management*, pages 13–18, 2013.
5. Gergely Alpár, Jaap-Henk Hoepman, and Johanneke Siljee. The identity crisis. security, privacy and usability issues in identity management. *CoRR*, abs/1101.0427, 2011.
6. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *ACM Conference on Computer and Communications Security*, pages 1087–1098, 2013.
7. E. Barker and J. Kelsey. NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators, 2012.
8. Lejla Batina, Jaap-Henk Hoepman, Bart Jacobs, Wojciech Mostowski, and Pim Vullers. Developing efficient blinded attribute certificates on smart cards via pairings. In *CARDIS*, pages 209–222, 2010.
9. 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.
10. Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.
11. BSI. TR-03110: Advanced Security Mechanisms for Machine Readable Travel Documents.
12. 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.
13. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials (extended version). *IACR Cryptology ePrint Archive*, 2010:496, 2010.
14. 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.
15. 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, 2001. Springer-Verlag.
16. 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.
17. Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO*, pages 413–430, 1999.
18. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.
19. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, pages 410–424, 1997.
20. Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, March 2005.
21. David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
22. Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security*, pages 63–86, 1998.
23. Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.
24. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

25. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30, 1997.
26. 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.
27. ISO/IEC. International standard 7816–4.
28. Anja Lehmann, Patrik Bichsel, Bud Bruegger, Jan Camenisch, Alberto Crespo Garcia, Thomas Gross, Andre Gutwirth, Moritz Horsch, Detlef Houdeau, Detlef Hühnlein, Frank-Michael Kamm, Stephan Krenn, Gregory Neven, Charles Bastos Rodriguez, Johannes Schmölz, and Charlotte Bolliger. Survey and analysis of existing eid and credential systems. Technical Report Deliverable D32.1, FutureID, 2013.
29. 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.
30. Eve Maler and Drummond Reed. The Venn of Identity: Options and Issues in Federated Identity Management. *IEEE Security and Privacy*, 6(2):16–23, March 2008.
31. 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.
32. Ingo Naumann and Giles Hogben. Privacy features of European eID card specifications. *Network Security*, 2008(8):9 – 13, 2008.
33. David Recordon and Drummond Reed. OpenID 2.0: A Platform for User-centric Identity Management. In *Proceedings of the Second ACM Workshop on Digital Identity Management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM.
34. Standards for Efficient Cryptography Group. Sec 2: Recommended elliptic curve domain parameters. SECG2, 2000.
35. 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.
36. 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*.

## Appendix A: Equality proofs of representation via prime-encoded attributes

The main operation of Idemix is the modular exponentiation where the number of these operations is related to the amount of the cardholder’s attributes that are hidden i.e.  $\mathcal{O}(l)$  for  $l$  attributes. Recently, Camenisch et al, proposed an alternative method for encoding attributes that reduces the overall number of modular exponentiations to 2 [13]. They utilize a base  $R_1$  for encoding all the attributes, which are represented as prime numbers. Therefore, the attribute corresponding to  $R_1$  consists of the product  $m_t = \prod_{i=1}^l m_i$  for  $l$  attributes. Proving the presence of an attribute  $m_i$  in  $m_t$  is performed via the coprime property: one shows that a certain attribute  $m_i$  can divide the product  $m_t$ . For instance, proving that the attribute  $m_1$  belongs to  $m_t$  is represented in zero knowledge as NIZK:  $\{(\varepsilon', \nu', \alpha_0, \alpha_1) : Z \equiv \pm R_0^{\alpha_0} (R_1^{m_1})^{\alpha_1} A^{\varepsilon'} S^{\nu'} \pmod{n}\}$ . In addition, the commitment  $C = Z^{m_t} S^r \pmod{n}$  and the  $t$ -values  $\tilde{C} = (Z^{m_1})^{\tilde{m}_h} S^r \pmod{n}$  and  $\tilde{C}_0 = Z^{\tilde{m}_t} S^{\tilde{r}} \pmod{n}$  must be computed, where  $\tilde{m}_h = m_t/m_r$  and  $m_r$  consists on the product of attributes  $m_i$  that are revealed (in this case  $m_r = m_1$ ). Moreover,  $Z, S \in QR_n$  are both part of the issuer public key as described in Section 3. Finally, the verifier checks  $C$  and  $C_0$  as  $\tilde{C} \stackrel{?}{=} C^{-c} (Z^{m_r})^{\tilde{m}_h} S^{\tilde{r}} \pmod{n}$  and  $\tilde{C}_0 \stackrel{?}{=} C^{-c} Z^{\tilde{m}_t} S^{\tilde{r}} \pmod{n}$  together with the verification of the ownership of the CL signature as described in Protocol 1. This is performed

using the following  $s\_values$  computed and sent by the card:  $\hat{m}_0 = \tilde{m}_0 + cm_0$ ,  $\hat{m} = \tilde{m} + cm$ ,  $\hat{m}_h = \tilde{m}_h + cm_h$  and  $\hat{r} = \tilde{r} + cr$ . In this case, the PRNG would compute the following sequence:  $init_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow \tilde{m}_h \Rightarrow \tilde{r} \Rightarrow r \Rightarrow \tilde{m}_t \Rightarrow reset_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow \tilde{m}_h \Rightarrow \tilde{r}$ . Otherwise, not revealing any attribute, that is, only proving the ownership of the signature would be represented as NIZK:  $\{(\varepsilon', \nu', \alpha_0, \alpha_1) : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} A^{\varepsilon'} S^{\nu'} \pmod{n}\}$ . This requires two exponentiations with independence of the number of attributes hidden. In this case, the PRNG would follow the same sequence depicted in Section 4.

Table 7: Performance overhead while verifying five attributes using a custom PRNG for generating the  $\hat{m}$  values (ms)

Work	Encoding	Case	$T_{sel\_cred}$	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(m_0, \dots, m_5   C, \hat{r}, \hat{m}_h)$	Total
This work	prime	BC	103.13	1,250.10	15.40, 11.61, 19.67	38.31, 11.53   13.94, 32.97, 51.14	1,547.83
This work	prime	WC	103.71	987.10	15.54, 11.44, 19.62	38.30*2	<b>1,214.24</b>

As depicted in Table 7 we notice that what we considered the worst case for normal encoding is the opposite here. We reuse the notation utilized in Sections 4–7 i.e. WC for hiding all the attributes and BC for revealing the content of a credential with the exception of  $m_0$ . Hence, only proving the ownership of a CL signature over a set of attributes without revealing any only requires 1,214.24 ms. In contrast, revealing all the attributes requires the computation of  $C$ ,  $\tilde{C}$  and  $\tilde{C}_0$ . Thanks to the optimizations carried out in Section 4 we can store  $C$  in RAM to avoid its recomputing when the verifiers requests its value. However, revealing all the attributes requires 492.20 ms more in comparison to the utilization of traditional encoding due to the additional modular exponentiations and multiplications required by the generation of  $C$ ,  $\tilde{C}$  and  $\tilde{C}_0$ .

We have recomputed the performance of standard and domain pseudonyms from Table 3 in Table 8 relying on prime-encoded attributes. In this respect, all the performance figures concerning the worst cases were improved i.e. 485.89 ms (standard pseudonyms) and 499.76 ms (domain pseudonyms in combination with standard pseudonyms, Section 5). 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 (Table 3).

Table 8: Performance analysis of proving the ownership of standard and domain pseudonyms (prime encoding, ms)

Implementation	Case	$T_{sel\_cred}$	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(m_0, \dots, m_5   C, \hat{r}, \hat{m}_h   nym, \hat{r}   dNym)$	Total
Nym	BC	103.13	1,720.73	15.41, 11.23, 19.46	38.42, 11.55   14.13, 32.48, 52.17   13.98, 50.33	2,083.01
Nym	WC	104.11	1,288.88	15.54, 11.64, 19.22	38.11*2   –   13.99, 50.86	<b>1,579.53</b>
Nym $\wedge$ dNym	BC	103.17	2,255.31	15.29, 11.51, 19.12	38.39, 11.83   14.17, 32.44, 52.05   14.16, 58.34   51.12	2,676.92
Nym $\wedge$ dNym	WC	104.33	1,487.86	15.22, 11.49, 19.48	38.02*2   –   13.58, 58.11   52.12	<b>1,838.25</b>

It can be possible to rely on prime encoding attributes for performing equality proofs with a better performance in comparison to the first two alternatives. In this respect, while the performance of the best case would be slightly worst due to the computation of the extra commitments, it can be possible to improve the performance of the worst one by reducing the number of exponentiations to  $\mathcal{O}(1+1)$  per credential instead of  $\mathcal{O}(5)$  per credential as in the alternatives a and b (Table 5). Proving that 2 credentials share  $m_0$  without revealing any attributes would be represented in zero knowledge as NIZK:  $\{(\varepsilon'_1, \nu'_1, \varepsilon'_2, \nu'_2, \mu, \alpha_1, \alpha_2) : Z^{(1)} \equiv \pm R_0^{(1)\mu} R_1^{(1)\alpha_1} A^{(1)\varepsilon'_1} S^{(1)\nu'_1}$

$\text{mod } n_1 \wedge Z^{(2)} \equiv \pm R_0^{(2)\mu} R_1^{(2)\beta_1} A^{(2)\varepsilon'_2} S^{(2)\nu'_2} \text{ mod } n_2\}$ . We note that there is an improvement of 496.59 ms and 989.82 ms in comparison to the alternatives b and a respectively (Table 9)

However, the computation of  $C$ ,  $\tilde{C}_o$  and  $\tilde{C}$  together with the two extra  $s\_values$  undermines any possibility of improving the figures related to the best cases from b and a.

Table 9: Performance overhead while verifying two credentials with 5 attributes using the equality proof (ms)

Alternative	Case	$T_{sel\_cred}$ (ms)	$T_{gen\_commit}$	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(m_0, \dots, m_5   C, \hat{r}, \hat{m}_h)$	Total
c	BC	103.23	3,145.11	(232.69, 91.37, 27.72) <sup>(1,2)</sup>	(32.11, 11.01) <sup>(1)</sup> , (11.14) <sup>(2)</sup>   (13.90, 32.95, 51.14) <sup>(1,2)</sup>	4,202.74
c	WC	104.17	2,023.94	(232.61, 91.52, 27.62) <sup>(1,2)</sup>	(31.19*2) <sup>(1,2)</sup>	<b>2,894.01</b>

We have also estimated the time that requires computing equality proofs up to 8 credentials using the alternatives a and c (Figure 1). We consider 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.

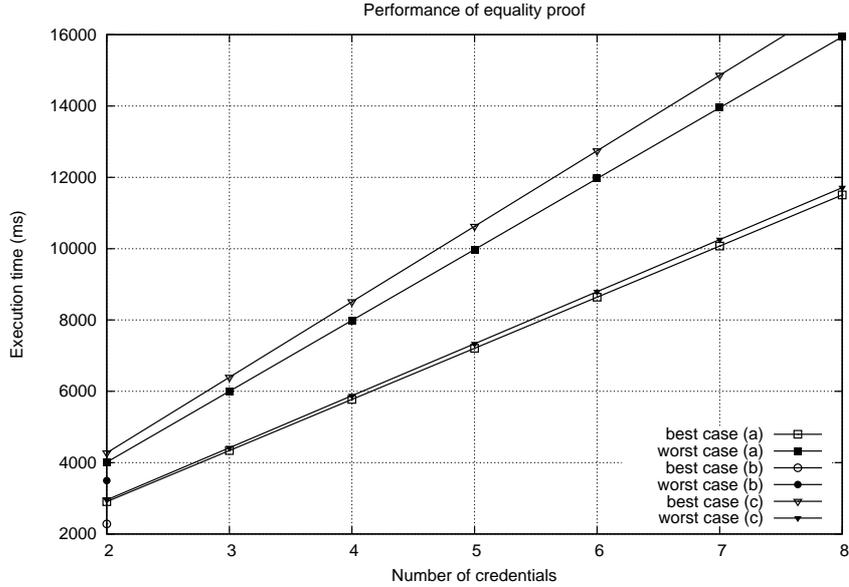


Fig. 1: Performance of the equality proof (credentials of five attributes)

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 an user should prove the ownership of  $n$  credentials without revealing her attributes.