Efficient Implementation of AND, OR and NOT Operators for ABCs

Antonio de la Piedra(✉)

ICIS DS, Radboud University Nijmegen, Nijmegen, The Netherlands
a.delapiedra@cs.ru.nl

Abstract. In the last few years several practitioners have proposed different strategies for implementing Attribute-based credentials (ABCs) on smart cards. ABCs allow citizens to prove certain properties about themselves without necessarily revealing their full identity. The Idemix ABC is the most versatile ABC system proposed in the literature, supporting peudonyms, equality proofs of representation, verifiable encryption of attributes and proving properties of attributes via AND, NOT and OR operators. Recently, Vullers et al. and De La Piedra et al. addressed the implementation of the selective disclosure operations, pseudonyms and multi-credential proofs such as equality proofs of representation. In this manuscript, we present implementation strategies for proving properties of user attributes via these operators and show how to combine them via external and internal commitment reordering.

Keywords: Attribute-based credentials · Smart cards

1 Introduction

Our everyday life is full of situations where we must identify ourselves. This is exemplified where we buy alcohol, cigarettes or other type of adult goods. In such process, we usually rely on our IDs in order to show that our age is consistent with the current legislation. However, in most of those operations we do not need to reveal our full identity. ABCs solve these privacy breaches by enabling users to reveal or hide the set of attributes that represent their identity according to the real need of the identification process. In so doing, the usual identification operation is replaced by an authorization according to the restricted set of attributes that are asked for. ABCs generally consist of a set of signed attributes that through certain cryptographic primitives can be used for authentication while tracing is avoided as well as ensuring that nobody can reuse the credential attributes or have access to them. Modern anonymous credential systems such as Idemix [8] and U-Prove [4] rely on blind and randomizable signatures in combination with proofs of knowledge [13]. While some practitioners have proposed several implementations of ABCs [3,19], the IRMA card[1] is the

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

M. Yung et al. (Eds.): INTRUST 2015, LNCS 9565, pp. 183–199, 2016.
DOI: 10.1007/978-3-319-31550-8_12

only open-source and practical implementation of Idemix. In this paper we rely on the current version of the card.

The main operation of ABCs is the selective disclosure. A user can reveal a reduced set of her attributes according to a presentation policy sent by a verifier [6]. However, in certain cases it can be useful to prove relations across attributes of the same credential. For instance, a restricted service can enforce an access control based on the ownership of an attribute a OR another one b. Moreover, it can ask if the cardholder is not owning a special type of attribute e.g. one that describes that her age is NOT higher than 18. All these operations are related to the AND, OR and NOT operators introduced by Camenisch et al. in [7]. In this manuscript, we address their implementation on constrained devices.

In the next section, we describe the work of other practitioners who implemented ABCs on smart cards and relate their performance figures with our work. In Sect. 3 we describe the main building blocks of ABCs. In Sect. 4, we sketch out the internals of the IRMA card. In Sects. 5 and 6 we present our strategies for executing complex proofs based on the AND, OR and NOT operators. We describe our results for combining them in Sect. 7. Finally, we end in Sect. 8 with some conclusions.

2 Related Work

Bichsel et al. [3] presented in 2009 the first implementation of Idemix based on Java Card (7.4 s, 1,280 bit RSA modulus) solely based on the selective disclosure of one attribute. These results preceded the design of Sterckx et al. [19] (4.2 s, 1,024 bit RSA modulus). Using the MULTOS platform, Vullers et al. presented an implementation of the issuing and verification operations of Idemix using credentials of 5 attributes (1–1.5 s, 1,024 bit RSA modulus). De La Piedra et al. [18] proposed the implementation of larger credentials, pseudonyms (1,486.51 ms) and multi-credential proofs (2,261.19 ms) on the same platform using a PRNG and variable reconstruction in RAM relying on the implementation of Vullers et al [18].

Contribution. In this manuscript, we present strategies for executing OR, NOT and AND operators over credentials based on prime-encoded attributes as described by [7]. Relying on the AND operator we found the limit of the amount of attributes we can issued in the target device[2] is 44. We always can perform this operation using attributes of different lengths in less than 2.7 s whereas issuing more than 5 attributes using traditional attributes requires more than 3 s [20]. This suggests that issuing prime-encoded attributes can be accompanied by the computation of pseudonyms. On the other hand, we observed that the

[2] Our performance figures have been extracted relying on a MULTOS ML3-R3-80K smart card using the SCM Microsystems SCL011 reader in a Intel Core i5-3230M CPU clocked at 2.60 GHz running Debian Linux 3.13.6-1, python 2.7.6, python-pyscard 1.6.12.1-4 and CHARM 0.43 [2].

verification of attributes using this operator is only optimal when none of the attributes are revealed (1.2 s, Sect. 7.1.3). We also present the implementation of the NOT operator using three approaches for solving the required Diophantine equation. Using credentials of the same size of the IRMA card we can perform the NOT operator in 1,974.96 ms with precomputatoin and between 2,016.41 and 2,135.53 ms using the extended Euclidean algorithm. We can perform the OR operator using the same type of credentials in 1,885.96 ms. Finally, we propose the internal and external reorganization of commitments for making it possible the combination of this operators: AND $\wedge$ NOT (2,201.90 ms), AND $\wedge$ OR (1,924.5 ms), NOT $\wedge$ OR (2,122.20 ms) and AND $\wedge$ NOT $\wedge$ OR (2,252.60 ms). By performing the proposed reorganizations of commitments we obtained reductions between 170.10 ms and 644.60 ms and between $9 \cdot 74$ and $1 \cdot 74$ bytes savings in RAM. Our results suggest that is actually possible to execute complex proofs of knowledge on embedded devices in reasonable times for on-line settings. Moreover, our performance figures are consistent with the current results in the literature [18,20].

3 Preliminaries

The IRMA card relies on a subset of the Idemix specification [20]. In this section, we describe the fundamentals of private ABCs and their main cryptographic blocks: non-interactive commitment schemes, blind signatures and zero-knowledge proofs. In this respect, we present the Camenisch-Lysyanskaya (CL) digital signature [10], the Fujisaki-Okamoto commitment scheme [16] and the Idemix ABC [8].

Non-interactive Commitment Schemes. These constructions are utilized in Idemix for committing to secret values during the issuing and verification operations. In so doing, one of the parties proves the knowledge of a committed value such as an attribute that is not revealed. Typically, a commitment scheme consists of two stages: commit and reveal i.e. a value x that is received as an input in the first stage will be revealed during the second one. Idemix relies on the Fujisaki-Okamoto commitment [16] scheme, which is statistically hiding and computationally binding when factoring is a hard problem. 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 $g^x h^r \mathrm{mod} n$.

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 [14]. In Idemix, the typical three

movement of sigma protocols (commitment, challenge and response[3]) is transformed into a Non Interactive Proof of Knowledge (NIZK) via the Fiat-Shamir heuristic [15] 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 [16].

The CL Digital Signature Scheme. The CL signature scheme is the main block of Idemix [10]. It provides multi-show unlinkability via the randomization of the issued signature. 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. In private ABCs systems [12], the users remain anonymous and are only known by their pseudonyms. They consist of organizations that issue and verify credentials so a user can prove its identity to a verifier while the issuer remains oblivious. In Idemix, this is performed via the multi-show unlinkability property of the CL digital signature scheme. Avoiding the transference of credentials between users is enforced 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 [9]) and verification (or VerifyCred [9]). 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 [5]), i.e. a list of attributes or conditions in a certain credential that must be fulfilled during an authentication operation[4].

[3] 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.

[4] For instance, an empty proof of possession over a set of attributes $(m_0, ..., m_5)$ is represented using the Camenisch-Staedler notation [11] as: 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.

4 The IRMA Card

IRMA supports up to 5 attributes by credential and relies on 1,204 special RSA modulus for performance reasons[5]. IRMA is based on the MULTOS card. Particularly, our 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.

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 hidden too. However, in IRMA the cardholder's attributes are never revealed to the issuer.

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 $n_1 \in_R \{0,1\}^{l_\emptyset}$ and sends it to the cardholder.
2. $C \rightarrow I$: The cardholder computes the commitment $U = S^{v'} R^{m_0} \mod n$ with $v' \in \pm\{0,1\}^{l_n + l_o}$. 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 $n_2 \in_R \{0,1\}^{l_o}$.
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 $\hat{Q} = A^e \mod n$.

[5] As described in [18], the attributes are represented as $l_m = 256$ bits. The rest of parameters are set as $l'_e = 120$ (size of the interval where the e values are selected), $l_\emptyset = 80$ (security parameter of the statistical ZKP), $l_H = 256$ (domain of the hash function in the Fiat-Shamir heuristic), $l_e = 504$ (size of e), $l_n = 1,024$ (size of the RSA modulus) and $l_v = 1,604$ bits (size of v).

[6] http://www.infineon.com/dgdl/SPO_SLE+78CX1280P_2012-07.pdf?folderId=db3a304325afd6e00126508d47f72f66&fileId=db3a30433fcce646013fe3d672214ab8 (Accessed 27 February 2015).

Issuing requires two NIZK (Table 1). 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 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 NIZK. 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 $(c,\ S_e)$.

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:

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

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 analyze this aspect in Sect. 5.1) 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.

Verification in IRMA. 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. Then, the card performs the operations depicted in Table 2 (`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. This set of states is mapped to the three moves described in Table 2.

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 $r_A \in_R \{0,1\}^{l_n + l_\emptyset}$ 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 $\tilde{e} \in_R \pm\{0,1\}^{l'_e + l_\emptyset + l_H}$, $\tilde{v}' \in_R \pm\{0,1\}^{l_v + l_\emptyset + l_H}$, and $\tilde{m}_i \in_R \pm\{0,1\}^{l_m + l_\emptyset + l_H} (i \in A_{\bar{r}})$ 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.

As described in [18] the latency of the verification operation can be modeled first according to the number of attributes per credential together with the number of attributes that are revealed (r) or hidden.

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

The time the PROVE_CREDENTIAL state requires is represented by T_{sel_cred}. Further, $T_{gen\_commit}(n, r)$ represents PROVE_COMMITMENT. Finally, $T_{get\_sig}(i)$ is related to the PROVE_SIGNATURE state whereas $T_{get\_attr}(i)$ represents the PROVE_ATTRIBUTE state.

We rely on the PRNG proposed by De La Piedra et al. in [18] for recomputing the associated pseudorandomness of the proofs. That approached made it possible to increase the number of attributes per credential by recomputing the $\tilde{m}_i$ values. In so doing, it is possible to generate the associated pseudorandomness during the generation of the t-values and obtain, on the fly, the same sequence while generating the s_values by resetting the PRNG as described in [18] e.g. $init_{PRNG}() \Rightarrow \tilde{m}_i \Rightarrow reset_{PRNG}() \Rightarrow \tilde{m}_i$.

5 Performance Evaluation of AND, OR and NOT Operators

The main operation of Idemix is the modular exponentiation. This operation is related to the number of attributes that a certain cardholder hides in an operation. In [7], Camenisch et al. proposed encoding the user attributes as prime numbers, reducing the overall number of modular exponentiations to 2. In so doing, they only utilize a base R_1 for encoding all the attributes as product $m_t = \prod_{i=1}^{l} m_i$ for l attributes. This encoding technique enables the possibility of performing selective disclosure (namely, using the AND operator), proving the absence of an attribute in a certain credential (NOT operator) and the possibility that one or more attributes are presents in m_t s.t. $R_1^{m_t}$ via the OR operator. This encoding technique is useful where the number of possible values in an attribute is restricted to only some e.g. masculine or feminine, and each possibility has associated a prime number.

We rely on the PRNG described in [18] for making it possible the execution of these proofs. Moreover, we introduce two techniques (internal and external commitment reorganizations) for reducing the amount of required exponentiations and the RAM required for storing the respective commitment in each step. External commitment reorganizations make it possible enabling the chaining of several proofs using the AND, NOT and OR operators in tandem. The internal reorganization of commitments means reordering the computations of commitments of a certain proof in order to save the computation time and the amount of utilized RAM.

5.1 The AND Operator

This operator performs the selective disclosure of these attributes by proving that a certain value m_i (which can be one attribute or a product of several ones) divides m_t. In this respect, proving that a certain attribute m_1 belongs to m_t is represented in zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_1) : Z \equiv \pm R_0^{\alpha_0}(R_1^{m_1})^{\alpha_1} A^{\varepsilon'} S^{\nu'} \bmod \mathrm{n}\}$. In addition, the commitments $C = Z^{m_t} S^r \bmod \mathrm{n}$, $\tilde{C} = (Z^{m_1})^{\tilde{m_h}} S^r \bmod n$ and $\tilde{C_0} = Z^{\tilde{m_t}} S^{\tilde{r}} \bmod \mathrm{n}$ must be computed, where $m_h = m_t/m_i$ and m_i consists of the product of attributes that are revealed (in this case $m_i = m_1$).

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: $\{(\varepsilon', \nu', \alpha_0, \alpha_1) : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} A^{\varepsilon'} S^{\nu'} \bmod \mathrm{n}\}$. This requires two exponentiations with independence of the number of attributes hidden.

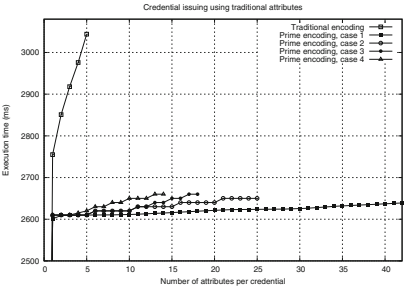
We can apply the internal organization of commitments. For instance, in the computation of the AND proofs we need to commit to the m_t value, i.e. the first attribute of the first base as $C = Z^{m_t} S^r \bmod \mathrm{n}$. However, the next commitment requires the computation of the S^r again as $\tilde{C} = (Z^{m_1})^{\tilde{m_h}}) S^r$. 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.

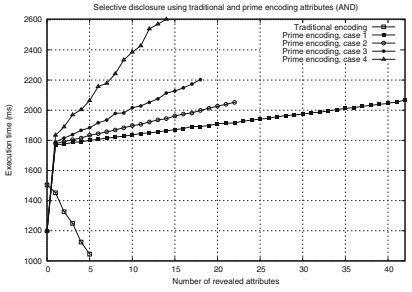
Issuing Prime-Encoded Attributes. Since the number of bases (and modular exponentiations) is reduced to the number of attributes in the credential to 2, we can compare the performance of the Idemix issuing operation using both prime-encoded and traditional attributes [20]. In this respect, it is expected that issuing prime-encoded attributes could reduce the latency associated to $T_{issuing}(n)$ as the number of attributes increase (Sect. 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. Hence, we can compare how the issuing operation in Idemix scales and observe how many attributes we can issue in the limit case of IRMA (5 attributes) [20].

We rely on the following methodology. First, we set a limit of 50 attributes per credential. Then, the only restriction is that $|m_t|$ cannot be greater than the $l_m = 256$ bit limitation according to the Idemix specification. Hence, we create 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 $l = 50 - 1$.

What we want to know is for each case, what is the maximum number of attributes per credential we can store according to l_m. Then, we create a list of primes according to its possibilities in each case when 50, 500, 5,000, 50,000



(a) Performance of issuing attributes via prime-encoded and traditional attributes



(b) Performance of selective disclosure using both types of encoding

Fig. 1. Issuing and verifying prime-encoded attributes

primes are involved. 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).

In relation to Fig. 1 (a), we have used the total number of bits for encoding the attributes for each case in order to obtain a fair comparison. As depicted, it is possible to not cross the 3 s margin of issuing traditional attributes and at the same time issue 44, 22, 18 and 14 attributes of different lengths under the 2.7 s limit. This could mean that it would be possible to also associated a pseudonym or several pseudonyms during the issuing of that credential and still maintain a decent performance in comparison to the utilization of traditional attributes but issuing from 3 to 8 times more attributes [20].

Verification of Prime-Encoded Attributes. 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. We take the limit case of IRMA (5 attributes) and compare it with verifying and hiding prime encoded attributes in Fig. 1 (b). 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. $R_0^{m_0} R_1^{m_t}$. Hence, only proving the ownership of a CL signature over a set of attributes without revealing any only requires 1,198.21 ms. In contrast, revealing all the attributes requires the computation of C, $\tilde{C}$ and $\tilde{C}_0$.

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 ($T_{gen\_commit}(n, r)$). 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 (Fig. 1 (b)). 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. $C, C_0, \tilde{C}$.

5.2 The NOT Operator

By using prime-encoded attributes it is possible to prove that an m_i attribute or set of attributes do not belong to m_t. This is done by showing that the integers x, y exists w.r.t. the following linear Diophantine equation $x \cdot m_t + y \cdot m_i = 1$.

We prove the ownership of a CL signature over m_0, m_t and the existence of (x, y) via zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_1, \chi, \upsilon, \rho, \rho') : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} A^{\varepsilon'} S^{\nu'} \bmod \mathrm{n} \ \wedge \mathrm{C} \equiv \pm \mathrm{Z}^{\alpha_1} \mathrm{S}^{\rho} \bmod \mathrm{n} \wedge \mathrm{Z} \equiv \pm \mathrm{C}^{\chi} (\mathrm{Z}^{\mathrm{m_i}})^{\upsilon} \mathrm{S}^{\rho'}\} \bmod \mathrm{n}$. The card must compute the commitments $C = Z^{m_t} S^r \bmod \mathrm{n}$, $\tilde{C} = C^{\tilde{x}} (Z^{m_i})^{\tilde{y}} S^{\tilde{r}'} \bmod \mathrm{n}$ and $\tilde{C}_c = Z^{\tilde{m}_t} S^{\tilde{r}} \bmod \mathrm{n}$ where $\tilde{r}$, $\tilde{r}'$, $\tilde{x}$,$\tilde{y}$ are randomizers [7].

In this case, the critical operation is the computation of the *(a, b)* pairs and how this operation scales for large primes. We propose two type of implementations: (1) precomputing the pairs (x, y) and (2) solving the Diophantine equation on the card. In the first case, given (x, y) for $x \cdot m_i + y \cdot m_t = 1$, m_t has always the same value and there are several combinations for m_i. Those possibilities can be stored in EEPROM if a small number of attributes is utilized. The number of (x, y) pairs that must be stored is related to the number of attributes per credential as $\sum_{i=1}^{l} c_n^i = \sum_{i=1}^{l} \binom{n}{i}$. Hence, for $l = 2$ attributes per credential, we would have to store three (x, y) pairs. In the case of 4 attributes, we store 15 pairs.

The second design is related to the computation of the Extended Euclidean algorithm on the smart card. We can use the instruction `PRIM_DIVIDEN` from the MULTOS specification that extracts the Euclidean division of two numbers i.e. q and r ($\mathcal{O}(ln^3\ N)$) in order to implement it. Other alternative is to use the binary GCD or Stein's algorithm [17]. This algorithm replaces the multiplications and divisions by bit-wise operations. Finally, the Lehmer's algorithm relies on the following idea ($\mathcal{O}(ln^2\ N)$). When a and b have the same size, the integer part w of the quotient a/b has one only digit. The goal is to find w while it is small and continue the involved operations via a matrix. The advantage of this method is that a large division is only done when needed, if w is greater than a certain base M.

In order to test the performance of these three algorithms, we have created four possible cases and have extracted performance figures in our target device. The attribute m_i can vary according to the number of attributes that are proved that are not in m_t. Its length will be greater according to the number of possibilities for each credential. In order to obtain an estimation of the computation time of each method on the MULTOS card we take 4 cases. If we take the first 10,000 primes, the numbers consist of 2 to 104,729. We can encode these values using 1 byte to 3 bytes (e.g. 0×019919 in the case of 104,729). However, m_i and m_t can increase according to all the possibilities an attribute can represent together with the number of involved attributes. We take four cases for an implementation based on 5 attributes (m_t) with different possibilities[7]. We rely on credentials of 5 attributes in this case in order to compare the performance of this operation with the selective disclosure via traditional attributes of the IRMA card [20].

[7] 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).

Table 3. 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**

The first aspect we notice from Table 3 is that the Stein's variant obtained the worst computational figures for the cases proposed despite it is been based on bit-wise operations, that are suppose to require less time as claimed by Akhavi et al. [1]. This is, however not true for MULTOS. For the maximum length of case 4 (9 bytes), we have measured the latency of all the operations involved: 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). These latencies make this option ill-suited when replacing the Euclidean division by operations that are suppose to require less cycles. On one hand, the Stein's variant requires more control operations and branches and on the other one, bit-wise operations have a similar latency than the Euclidean division. Due to the proprietary nature of the SLE 78CX1280P chip we cannot claim that the Euclidean division is being performed via the hardware accelerator of the target device. Moreover, since MULTOS is based on MEL byte code that is executed in a virtual machine, we cannot be sure that code optimizations (written in C) can result in any speed up. Finally, we are unaware of any side channel analysis (SCA) countermeasures implemented on the card, but there is a possibility that the designers wanted to homogenize the latency of a group of simple arithmetic operations in order to make them indistinguishable.

In the case of the Lehmer's variant, for single-precision values of 32 bits or less, 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. Moreover, in our target device is not possible to tune the precision and adjust the assembler code since that is then translated into byte codes, executed by the virtual machine.

We have depicted in Table 4[8] the performance figures of the NOT operator for each case. In the precomputation strategy we only show the first case since increasing the length of the operand does not alter the result significantly. On the other hand, the computation of the pairs (x, y) is performed during the

[8] We use the following notation in Tables 4, 5 and 6: PRE means precomputation, 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.

Table 4. Performance figures of the NOT operator while precomputing the (x, y) pair and relying on the Euclidean algorithm (ms)

Case	T_{sel_cred}	T_{gen_commit}	$T_{get\_sig}(A, e, v)$	$T_{get\_attr}(\hat{m}_0, \hat{m}_t \vert \hat{r}, \hat{r}', \hat{a}, \hat{b}, C)$	Total
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	15.677, 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

computation of $\hat{a}$, adding its latency to $T_{get\_attr}(i)$ (Sect. 2). Thanks to the low latency operation of the PRIM_DIVIDEN primitive we obtained latencies between 2,015.41 and 2,135.35 ms.

5.3 The OR Operator

The utilization of this operator enables the cardholder to prove that an attribute m_i or more attributes encoded as a product can be found in m_t s.t. $m_t = \prod_{i=0}^{l}$ w.r.t. $R_1^{m_t}$. In so doing, we rely on the following fact: given an attribute $m_i \in m_t$, an integer x exists s.t. $x \cdot m_i = \prod_{i=1}^{l} m_i = m_t$. This is proved in zero knowledge as NIZK: $\{(\varepsilon', \nu', \alpha_0, \alpha_1, \chi, \rho, \rho') : Z \equiv \pm R_0^{\alpha_0} R_1^{\alpha_1} A^{\varepsilon'} S^{\nu'} \bmod \mathrm{n} \wedge \mathrm{C} \equiv \pm \mathrm{Z}^{\alpha_1} \mathrm{S}^{\rho} \bmod \mathrm{n} \wedge \mathrm{C}' \equiv \pm \mathrm{C}^{\chi} \mathrm{S}^{\rho'}\} \bmod \mathrm{n}$[9]. The card must compute three commitments $C = Z^{m_t} \cdot S^r \bmod \mathrm{n}, \tilde{\mathrm{C}} = \mathrm{Z}^{\tilde{\mathrm{m}}_t} \cdot \mathrm{S}^{\tilde{\mathrm{r}}} \bmod \mathrm{n}, \tilde{\mathrm{T}} = \mathrm{C}^{\tilde{\mathrm{x}}} \cdot \mathrm{S}^{\tilde{\mathrm{r}}_1}$ 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 first obstacle for implementing this primitive was to over come the lack of support of signed arithmetic on the card. This means creating wrappers over the multiplication and addition operations supporting sign extensions due to the computation of $r_1 = r_0 \cdot x$. Afterwards, the operation is performed in two's complement. By using the RAM reductions achieved thanks to the PRNG described in [18] and 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.

We have depicted in Table 4 the performance figures of case 1, described in Sect. 5.1. We can compute this operation withing 1,885.96 ms. Since this operation scales with m_i at the same pace of the AND, OR operators without the m_t product. Since the commitments utilized only involved two multi modular exponentiations, we can obtain a reduction of 1,974.96 - 1,885.96 (89) ms in comparison to the NOT operation.

[9] In this manuscript we only address the first version of this NIZK described in [7] and leave the second one beyond the scope of this work due the computation limitations of our target device.

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

Combination	Cases	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

5.4 Combination of Operators for Prime-Encoded Credentials

It can be useful to prove certain properties of a prime-encoded credential by utilizing a group of these operators. For instance, one could prove that an attribute a is in m_t s.t. $R_1^{m_t}$, b is NOT AND c OR d could be present. In so doing, it can be possible to perform some degree of commitment reorganization (i.e. external reorganization) in order to optimize the computation of the required commitments and response values.

Given the AND, NOT and OR operators, we consider the following combinations in order to obtain the best combination and estimate its performance. First, we discuss 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 (Sect. 5.2). 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. Then, in AND $\wedge$ OR, the OR operator (Sect. 5.3) 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 one time

Table 6. RAM savings by recomputing the pseudorandomnes in each primitive

Operator	PRNG sequence	No. of commitments	Required RAM (bytes)	RAM saved (bytes)
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	222	148
AND HA	$\tilde{m}_t, \tilde{m}_0 \vert \tilde{m}_t, \tilde{m}_0$	1	74	-
AND $\wedge$ Nym RA	$\tilde{m}_0, \tilde{m}_h, \tilde{r}, \tilde{r'}, r, \tilde{m}_t \vert [\_], \tilde{m}_0, \tilde{m}_h, \tilde{r'}, \tilde{r}, [\_], [\_], \tilde{m}_t$	5	370	296
AND $\wedge$ Nym HA	$\tilde{m}_t, \tilde{m}_0, \tilde{r'} \vert \tilde{m}_t, \tilde{m}_0, r'$	2	148	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	444	370
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	370	296
AND RA $\wedge$ 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	370	296
AND HA $\wedge$ 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	296	222
AND RA $\wedge$ 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	444	370
AND HA $\wedge$ 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	370	296
NOT $\wedge$ OR	$\tilde{b}, \tilde{a}, \tilde{r'}, \tilde{m}_t, \tilde{r}, \tilde{m}_0, \tilde{r'}, \tilde{r}, \tilde{r}_1, \tilde{x} \vert \tilde{b}, \tilde{a}, \tilde{r'}, \tilde{m}_t, \tilde{r}, \tilde{m}_0, \tilde{r'}, [\_], \tilde{r}_1, \tilde{x}$	5	666	592
AND RA $\wedge$ NOT $\wedge$ OR	$\tilde{m}_0, \tilde{m}_h, r, \tilde{r'}, \tilde{m}_t, \tilde{b}, \tilde{a}, \tilde{r}_1, \tilde{x} \vert \tilde{m}_0, \tilde{m}_h, [\_], [\_], \tilde{m}_t, \tilde{b}, \tilde{a}, \tilde{r}_1, \tilde{x}$	5	740	666
AND HA $\wedge$ NOT $\wedge$ OR	$\tilde{b}, \tilde{a}, \tilde{r'}, \tilde{m}_t, \tilde{r}, \tilde{m}_0, \tilde{r'}, \tilde{r}, \tilde{r}_1, \tilde{x} \vert \tilde{b}, \tilde{a}, \tilde{r'}, \tilde{m}_t, \tilde{r}, \tilde{m}_0, \tilde{r'}, [\_], \tilde{r}_1, \tilde{x}$	5	666	592

when combined and the AND proof can be executed with the optimizations discussed in Sect. 5. In the case of NOT $\wedge$ OR, 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. Finally, AND $\wedge$ NOT $\wedge$ OR. 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 the AND operator can be executed with internal commitment reorganization.

By performing external commitment reorganization we can obtain reductions in performance between 170.10 ms and 644.60 ms (Table 5) as well as in RAM (Table 6). This is mainly achieved where the three types of operators are being used and the commitments $C, \tilde{C}$ are reused together with the randomizers recomputed by the PRNG described in [18]. We rely on 5 attributes and on the cases created for the NOT operator (Sect. 5.2) together with the option where AND has the worst performance i.e. revealing all the attributes.

6 Conclusions

In this manuscript we have presented different strategies for implementing the operators for prime-encoded attributes described in [7]. We showed that when the number of attributes is large it can be possible to rely on prime-encoded proofs for improving the issuing process. Moreover, this also applies to the verification of a considerable amount of attributes. Besides, the selective disclosure operation can be improved in cases where hiding is needed by relying on prime-encoded attributes. Moreover, 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. 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.

References

1. Akhavi, A., Vallée, B.: Average Bit-Complexity of Euclidean Algorithms. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 373–387. Springer, Heidelberg (2000)
2. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. J. Crypt. Eng. **3**(2), 111–128 (2013)
3. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard Java Card. In: ACM Conference on Computer and Communications Security, pp. 600–610 (2009)
4. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge (2000)
5. Camenisch, J., Dubovitskaya, M., Enderlein, R.R., Lehmann, A., Neven, G., Paquin, C., Preiss, F.-S.: Concepts and languages for privacy-preserving attribute-based authentication. J. Inf. Sec. Appl. **19**(1), 25–44 (2014)
6. Camenisch, J., Dubovitskaya, M., Lehmann, A., Neven, G., Paquin, C., Preiss, F.-S.: Concepts and languages for privacy-preserving attribute-based authentication. In: Fischer-Hübner, S., de Leeuw, E., Mitchell, C. (eds.) IDMAN 2013. IFIP AICT, vol. 396, pp. 34–52. Springer, Heidelberg (2013)
7. Camenisch, J., Groß, T.: Efficient attributes for anonymous credentials (extended version). IACR Cryptol. ePrint Arch. **2010**, 496 (2010)
8. Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In: ACM Conference on Computer and Communications Security, pp. 21–30 (2002)
9. Camenisch, J.L., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 93. Springer, Heidelberg (2001)
10. Camenisch, J.L., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
11. Camenisch, J.L., Stadler, M.A.: Efficient group signature schemes for large groups. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
12. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985)
13. Damgård, I.B.: Commitment schemes and zero-knowledge protocols. In: Damgård, I.B. (ed.) EEF School 1998. LNCS, vol. 1561, p. 63. Springer, Heidelberg (1999)
14. Damgård, I.B.: Efficient concurrent zero-knowledge in the auxiliary string model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (2000)
15. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
16. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)
17. Knuth, D.E.: The Art of Computer Programming, Volume II: Seminumerical Algorithms, vol. 2, 2nd edn. Addison-Wesley, Boston (1981)

18. de la Piedra, A., Hoepman, J.-H., Vullers, P.: Towards a full-featured implementation of attribute based credentials on smart cards. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 270–289. Springer, Heidelberg (2014)
19. Sterckx, M., Gierlichs, B., Preneel, B., Verbauwhede, I.: Efficient implementation of anonymous credentials on java card smart cards. In: 1st IEEE International Workshop on Information Forensics and Security (WIFS), pp. 106–110. IEEE, London, UK, 2009 (2009)
20. Vullers, P., Alpár, G.: Efficient selective disclosure on smart cards using idemix. In: Fischer-Hübner, S., de Leeuw, E., Mitchell, C. (eds.) IDMAN 2013. IFIP AICT, vol. 396, pp. 53–67. Springer, Heidelberg (2013)