I Reveal My Attributes

IRMA card

Technical Specification

Version 0.1

ir. Pim Vullers

Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.
p.vullers@cs.ru.nl

February 26, 2013

Part I

APDU Communication

Chapter 1

Introduction

This part of the technical specifications contains all necessary details about the APDU communication between a terminal and an IRMA card.

We can distinguish a few different categories of communication. First there are the basic cryptographic operations of the Idemix technology provided by the card. These are exposed to a terminal by a set of APDU commands described in Chapter 3. Besides these basic operations we have added a number of commands to initialise the card (Chapter 4) and to perform card management and maintaince (Chapter 5).

Chapter 2

Session

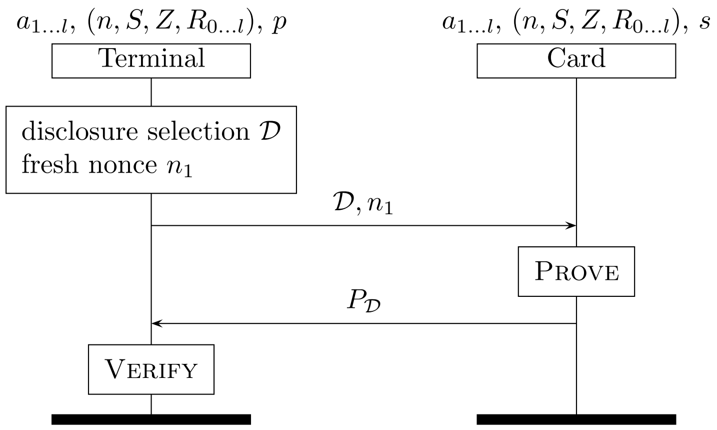

Figure 2.1: Overview of a session with an IRMA card.

2.1 Application Selection

Table 2.1: Command: SELECT_APPLICATION

CLA	0x00
INS	0xA4
P1	0x04
P2	0x00
LC	size of *AID*
DATA	*AID*

Table 2.2: Response: SELECT_APPLICATION

DATA	-
SW	0x9000 (if the application has been selected), or 0x6A82 (if the application does not exist)

2.2 Session Setup

2.3 Secure Messaging

The IRMA card uses the secure messaging scheme specified in [BSI TR-03110 - Part 3, Appendix E] which is also used by the electronic passports.

2.4 PIN

2.4.1 Verification

Table 2.3: Command: VERIFY_PIN

CLA	0x00
INS	0x20
P1	0x00
P2	0x00 (for credential PIN), or 0x01 (for management PIN)
LC	8
DATA	*PIN code* (possibly padded)

Table 2.4: Response: VERIFY_PIN

DATA	-
SW	0x9000 (if the PIN is valid), or 0x63cx (if the PIN is invalid, x tries remaining), or 0x6700 (if the length is not correct), or 0x6B00 (if the PIN does not exist)

2.4.2 Modification

Table 2.5: Command: MODIFY_PIN

CLA	0x80
INS	0x10
P1	0x00
P2	0x00 (for credential PIN), or 0x01 (for management PIN)
LC	16
DATA	old *PIN code* (possibly padded), new *PIN code* (possibly padded)

Table 2.6: Response: MODIFY_PIN

DATA	-
SW	0x9000 (if no error occurred), or 0x63cx (if the PIN is invalid, x tries remaining), or 0x6700 (if the length is not correct), or 0x6B00 (if the PIN does not exist)

Chapter 3

Idemix

3.1 Issuance

3.1.1 Protocol Initialisation

Start Issuance

To initiate a new issuance protocol the card first has to be put in issuance mode. This command is used to start a new issuance session for the credential specified by its parameters. The *credential ID* is used by the card to identify the credentials it contains. If the credential specified by this command already exists the session will be aborted. Furthermore this command is used to set the *context* information for this session. This information is used to link generated challenges to this session.

Table 3.1: Command: ISSUE_CREDENTIAL

CLA	0x80
INS	0x10
P1	*Credential ID* (most significant byte)
P2	*Credential ID* (least significant byte)
LC	ℓ_H
DATA	*Context*

Table 3.2: Response: ISSUE_CREDENTIAL

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6900 (if the credential cannot be issued), or 0x6982 (if the security status is not satisfied), or 0x6986 (if the credential already exists), or 0x6B00 (if the credential ID is not allowed)

After the command has been successfully executed, the card is put in issuance mode and the specified credential (identified by its ID) has been selected.

Public Key

Before the issuance protocol can be executed the public key of the issuer has to be set. This key consists of several elements: a modulus (n), a generator (S), a fixed value (Z) and a number of bases (R_i). Each of these elements requires its own invocation of this command, with the correct parameters, to set the value on the card. More information about the public key and its elements can be found in the cryptography part of this specification (Part II).

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.3: Command: PUBLIC_KEY

CLA	0x80
INS	0x11
P1	0x00 (for n), or 0x01 (for S), or 0x02 (for Z), or 0x03 (for R_i)
P2	0x00 (for n, S and Z), or index i (for R_i)
LC	ℓ_n
DATA	n, or S, or Z, or R_i

Table 3.4: Response: PUBLIC_KEY

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode), or 0x6986 (if the element has already been initialised), or 0x6B00 (if the element does not exist)

Attributes

Before the issuance protocol can be executed the actual attributes that will be issued have to be set. While these attributes a_i might have different lengths they should be padded to meet the fixed length for attributes as specified in the configuration details (Section 5.1).

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.5: Command: ATTRIBUTES

CLA	0x80
INS	0x12
P1	0x00
P2	index i (for a_i)
LC	ℓ_m
DATA	a_i

Table 3.6: Response: ATTRIBUTES

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode), or 0x6986 (if the attribute has already been initialised), or 0x6A80 (if the attribute has a null value), or 0x6B00 (if the attribute does not exist)

3.1.2 Protocol Execution

Card Commitment

The issuance protocol starts with a commitment U to the secret key and the blinding value to be used for the new credential. To prove that this commitment is computed correctlty the card also generates a non-interactive proof of correctness using the nonce n_1 provided by the issuer.

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.7: Command: ISSUE_COMMITMENT

CLA	0x80
INS	0x1A
P1	0x00
P2	0x00
LC	$\ell_{\o}$
DATA	n_1

Table 3.8: Response: ISSUE_COMMITMENT

DATA	U (ℓ_n bytes)
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode), or 0x6986 (if the nonce has already been initialised)

Proof of Correctness for U

During the execution on the previous command a non-interactive proof of corectness has been generated for U. This proof consists of a challenge c and the responses $\hat{v}'$ and $\hat{s}$. Using this command the issuer can retrieve this values in order to verify the correctness of the commitment.

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.9: Command: COMMITMENT_PROOF

CLA	0x80
INS	0x1B
P1	0x00 (for c), or 0x01 (for $\hat{v}'$), or 0x02 (for $\hat{s}$)
P2	0x00
LC	ℓ_H (for c), or $\ell_{\hat{v}'}$ (for $\hat{v}'$), or $\ell_{\hat{s}}$ (for s_A)
DATA	c, or $\hat{v}'$, or s_A

Table 3.10: Response: COMMITMENT_PROOF

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode), or 0x6986 (if the element has already been initialised), or 0x6B00 (if the element does not exist)

Card Challenge

In the next protocol step the issuer will create the signature for the credential and generate a non-interactive proof of correctness for this signature. In order to guarantee freshness of this proof the issuer first requests a fresh nonce n_2 from the card using this command.

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.11: Command: CHALLENGE

CLA	0x80
INS	0x1C
P1	0x00
P2	0x00
LC	-
DATA	-

Table 3.12: Response: CHALLENGE

DATA	n_2 ($\ell_\varnothing$ bytes)
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode), or 0x6986 (if the nonce has already been generated)

Credential Signature

The issuer constructs a Camenisch-Lysyanskaya blind signature over the attributes and the cards secret key (using the commitment). This command is used to send the partial signature (A, e, v'') to the card. To prove that this signature is computed correctlty the card also generates a non-interactive proof of correctness using the nonce n_2 provided by the card.

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.13: Command: ISSUE_SIGNATURE

CLA	0x80
INS	0x1D
P1	0x00 (for A), or 0x01 (for e), or 0x02 (for v''), or 0x03 (for verification)
P2	0x00
LC	ℓ_n (for A), or ℓ_e (for e), or ℓ_v (for v''), or - (for verification)
DATA	A, or e, or v'', or - (for verification)

Table 3.14: Response: ISSUE_SIGNATURE

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode), or 0x6986 (if the element has already been initialised), or 0x6B00 (if the element does not exist)

Proof of Correctness for A

During the execution on the previous command a non-interactive proof of corectness has been generated for (A, e, v''). This proof consists of a challenge c and the response s_e. Using this command the issuer sends these values such that the card can verify the correctness of the signature.

Note that this command can only be executed in issuance mode, hence it should be send after a successful invocation of the ISSUE_CREDENTIAL command.

Table 3.15: Command: SIGNATURE_PROOF

CLA	0x80
INS	0x1E
P1	0x00 (for c), or 0x01 (for s_e), or 0x02 (for verification)
P2	0x00
LC	ℓ_H (for c), or ℓ_n (for s_e), or - (for verification)
DATA	c, or s_e, or - (for verification)

Table 3.16: Response: SIGNATURE_PROOF

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in issuance mode or proof invalid), or 0x6986 (if the element has already been initialised), or 0x6B00 (if the element does not exist)

3.2 Proof Generation (Selective Disclosure)

3.2.1 Protocol Initialisation

Start Proof

To initiate a new proving protocol the card first has to be put in proving mode. This command is used to start a new proving session for the credential specified by its parameters. The *credential ID* is used by the card to identify the credentials it contains. If the credential specified by this command does not exist the session will be aborted. Furthermore this command is used to set the *context* information for this session. This information is used to link generated challenges to this session.

Table 3.17: Command: PROVE_CREDENTIAL

CLA	0x80
INS	0x20
P1	*Credential ID* (most significant byte)
P2	*Credential ID* (least significant byte)
LC	ℓ_H
DATA	*Context*

Table 3.18: Response: PROVE_CREDENTIAL

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6A88 (if the credential does not exist), or 0x6B00 (if the credential ID is not allowed)

After the command has been successfully executed, the card is put in proving mode and the specified credential (identified by its ID) has been selected.

Disclosure Selection

Before the issuance protocol can be executed the disclosure selection has to be set. This *selection* is encoded as a bitmask (of 16 bits): if the i-th bit of the selection is 1, attribute a_i should be disclosed.

Note that this command can only be executed in proving mode, hence it should be send after a successful invocation of the PROVE_CREDENTIAL command.

Table 3.19: Command: SELECTION

CLA	0x80
INS	0x21
P1	*Selection* (most significant byte)
P2	*Selection* (least significant byte)
LC	-
DATA	-

Table 3.20: Response: SELECTION

DATA	-
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in proving mode), or 0x6986 (if the selection has already been initialised), or 0x6A80 (if the selection is invalid), or 0x6A88 (if the selected attribute does not exist)

3.2.2 Protocol Execution

Card Commitment

This protocol basically only involves the generation of a non-interactive zero-knowledge proof. To guarantee the freshes of this proof the verifier sends a nonce n_1 that gets included in the challenge c which is a commitment by the card according to the Fiat-Shamir heuristic.

Note that this command can only be executed in proving mode, hence it should be send after a successful invocation of the PROVE_CREDENTIAL command.

Table 3.21: Command: PROVE_COMMITMENT

CLA	0x80
INS	0x2A
P1	0x00
P2	0x00
LC	$\ell_{\varnothing}$
DATA	n_1

Table 3.22: Response: PROVE_COMMITMENT

DATA	c (ℓ_H bytes)
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in proving mode)

Credential Signature

Once the verifier has received the commitment the verifier can retrieve the randomised signature $(A', \hat{e}, \hat{v})$ over the attributes.

Note that this command can only be executed in proving mode, hence it should be send after a successful invocation of the PROVE_CREDENTIAL command.

Table 3.23: Command: PROVE_SIGNATURE

CLA	0x80
INS	0x2B
P1	0x00 (for A'), or 0x01 (for $\hat{e}$), or 0x02 (for $\hat{v}$))
P2	0x00
LC	-
DATA	-

Table 3.24: Response: PROVE_SIGNATURE

DATA	A' (ℓ_n bytes), or $\hat{e}$ ($\ell_{\hat{e}}$ bytes), or $\hat{v}$ ($\ell_{\hat{v}}$ bytes)
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in proving mode), or 0x6B00 (if the element does not exist)

Response values

The last ingredient needed to verify the proof are the response value, which are either the disclosed attributes $a_{i \in \mathcal{D}}$ (retrieved using the ATTRIBUTE command) or blinded responses $\hat{a}_{i \notin \mathcal{D}}$ for the attributes that are not disclosed (retrieved using the RESPONSE command).

Note that these commands can only be executed in proving mode, hence they should be send after a successful invocation of the PROVE_CREDENTIAL command.

Table 3.25: Command: ATTRIBUTE

CLA	0x80
INS	0x2C
P1	0x00
P2	index i (for a_i)
LC	-
DATA	-

Table 3.26: Response: ATTRIBUTE

DATA	a_i (ℓ_m bytes)
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in proving mode or the attribute is not disclosed), or 0x6B00 (if the attribute does not exist)

Table 3.27: Command: RESPONSE

CLA	0x80
INS	0x2D
P1	0x00
P2	index i (for $\hat{a}_i$)
LC	-
DATA	-

Table 3.28: Response: RESPONSE

DATA	$\hat{a}_i$ ($\ell_{\hat{m}}$ bytes)
SW	0x9000 (if no error occurred), or 0x6700 (if the length is not correct), or 0x6982 (if the security status is not satisfied), or 0x6985 (if not in proving mode), or 0x6B00 (if the response does not exist)

Chapter 4

Card Initialisation

Chapter 5

Card Management

Part II

Cryptography

5.1 Configuration Details

Table 5.1: System parameters.

parameter	description	bits	bytes	integers[1]
ℓ_n	size of RSA modulus	1024	128	32
ℓ_e	size of RSA exponent	504	64	16
$\ell_{e'}$	size of the RSA exponent interval	120	16	4
ℓ_H	size of the hash output	160	20	5
ℓ_m	size of the attributes	256	32	8
ℓ_v	size of the blinding value	1604	201	51
$\ell_\varnothing$	security of the statistical zero-knowledge property	80	10	3
derived				
$\ell_{\hat{e}}$	$\ell_{e'} + \ell_\varnothing + \ell_H$	360	45	12
$\ell_{\hat{m}}$	$\ell_m + \ell_\varnothing + \ell_H$	496	62	16
$\ell_{\hat{s}}$	$\ell_m + \ell_\varnothing + \ell_H + 1$	497	63	16
$\ell_{\hat{v}}$	$\ell_v + \ell_\varnothing + \ell_H$	1844	231	58
$\ell_{\hat{v}'}$	$\ell_n + 2 \cdot \ell_\varnothing + \ell_H$	1344	168	42

5.2 Credential Issuance

This section describes the card operations in the issuance protocol.

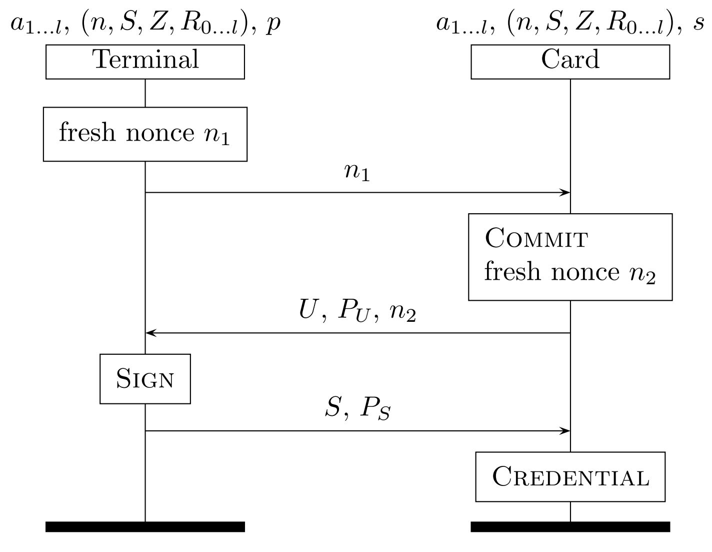

Figure 5.1: Idemix issuance protocol.

[1] 32-bit

5.2.1 Initialization

Important parameters and setups on the card in this protocol:

- number of attributes: l
- issuer public key: $K = (n, Z, S, R_{0\ldots l})$
- session specific context: `context`
- card's master secret key: s
- attributes: $a_{1\ldots l}$

5.2.2 Card Commitment

Algorithm 1 Construct the card's commitment (issuance protocol round 1).

Receives n_1
Stores v', n_2
Sends U, P_U, n_2

procedure COMMIT(n_1)
    $U, v' \leftarrow$ COMPUTECOMMITMENT( )
    $P_U \leftarrow$ PROVECOMMITMENT(U, v', n_1)

To start the card computes a commitment U which includes its secret key s and a fresh blinding value v' (Algorithm 2). Next, the card generates a non-interactive proof P_U (5.1) that this commitment was computed correctly (Algorithm 3).

The card stores v' for later use in this protocol and sends U, P_U and a fresh nonce to the terminal.

$$P_U = SPK\{\beta, \sigma : U \equiv \pm S^{\beta} \cdot R_0^{\sigma} \pmod n \wedge \sigma \in \pm\{0,1\}^{\ell_m + \ell_\varnothing + \ell_H + 1}\} \quad (5.1)$$

Algorithm 2 Compute the commitment U.

function COMPUTECOMMITMENT( )
    $v' \in_R \pm\{0,1\}^{\ell_n + \ell_\varnothing}$
    $U \leftarrow S^{v'} \cdot R_0^s \mod n$
    return U, v'

5.2.3 Construct Signature (terminal)

5.2.4 Construct Credential

After I verifies the computations from Round 1 in Round 2 and generates a temporary CL signature on $(m_1, \ldots, m_l)$, S verifies it, then it computes and stores the permanent signature.

To complete the credential the card computes v and verifies the signature (A, e, v) and its proof P_2, moreover, it stores the credential:

Algorithm 3 Generate the non-interactive proof of correctness P_U.

function ProveCommitment(U, v', n_1)
  $\tilde{s} \in_R \pm\{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+1}$
  $\tilde{v}' \in_R \pm\{0,1\}^{\ell_n+2\ell_\varnothing+\ell_H}$
  ▷ Commitment
  $\tilde{U} \leftarrow S^{\tilde{v}'} \cdot R_0^{\tilde{s}} \mod n$
  ▷ Challenge computed using Fiat-Shamir heuristic
  $c \leftarrow \mathcal{H}(\texttt{context}||U||\tilde{U}||n_1)$
  ▷ Responses (note: no modular reduction)
  $\hat{s} \leftarrow \tilde{s} + c \cdot s$
  $\hat{v}' \leftarrow \tilde{v}' + c \cdot v'$
  return $P_U = (c, \hat{v}', \hat{s})$

Algorithm 4 Construct the issuer's signature (issuance protocol round 2).

Receives U, P_U, n_2
Sends S, P_S

procedure Sign(U, P_U, n_2)
  if VerifyCommitment$(U, P_U, n_1) \neq$ Valid **then**
    Abort
  $S \leftarrow$ ComputeSignature( )
  $P_S \leftarrow$ ProveSignature(S, n_2)

Algorithm 5 Verify the (proof of) correctness of U.

function VerifyCommitmentProof$(U, (c, \hat{v}', \hat{s}), n_1)$
  if $\hat{v}' \notin \pm\{0,1\}^{\ell_m+2\ell_\varnothing+\ell_H+1}$ **then**
    return Invalid
  $\hat{U} \leftarrow U^c \cdot S^{\hat{v}'}$
  $\hat{c} \leftarrow \mathcal{H}(\texttt{context}||U||\hat{U}||n_1)$
  if $\hat{c} \neq c$ **then**
    return Invalid
  return Valid

Algorithm 6 Compute a CL signature on the attributes.

function ComputeSignature( )
  $e \in_R [2^{\ell_e 1}, 2^{\ell_e - 1} + 2^{\ell_{e'} - 1}]$
  $\tilde{v} \in_R \{0,1\}^{\ell_v - 1}$
  $v'' \leftarrow 2^{\ell_v - 1} + \tilde{v}$
  $Q \leftarrow \dfrac{Z}{U \cdot S^{v''} \cdot \prod_{i=1}^{l} R_i^{m_i}} \mod n$
  $A \leftarrow Q^{e^{-1} \mod p' \cdot q'} \mod n$
  return $S = (A, e, v'')$

Algorithm 7 Generate the non-interactive proof of correctness P_S.

function ProveSignature($(A, e, v), n_2$)
  $r \in_R \mathbb{Z}^*_{p' \cdot q'}$
  $\tilde{A} \leftarrow Q^r \mod n$
  $c \leftarrow \mathcal{H}(\texttt{context}||Q||A||n_2||\tilde{A})$
  $\hat{e} \leftarrow r - c \cdot e^{-1} \mod p' \cdot q'$
  return $P_S = (c, \hat{e})$

Algorithm 8 Contruct the credential (issuance protocol round 3).

Receives S, P_S
Retrieves v', n_2
Stores C

procedure Credential(S, P_S, v', n_2)
  if VerifySignature(S, P_S, n_2) $\neq$ Valid **then**
    Abort
  $C \leftarrow$ ComputeCredential(S, v')
  if VerifyCredential(C) $\neq$ Valid **then**
    Abort

Algorithm 9 Verify the (proof of) correctness of S.

function VerifySignature($(A, e, v), (c, \hat{e}), n_2$)
  $\hat{A} \leftarrow A^{c + \hat{e} \cdot e} \mod n$
  if $c \neq \mathcal{H}(\texttt{context}||Q||A||n_2||\hat{A})$ **then**
    return Invalid
  return Valid

Algorithm 10 Construct the final credential.

function computeCredential($(A, e, v''), v'$)
  $v \leftarrow v'' + v'$
  return $C = (a_{1 \ldots l}, A, e, v)$

Algorithm 11 Verify the constructed credential S

function VERIFYCREDENTIAL($(a_{1\ldots l}, A, e, v)$)
  if $e \notin prime$ **then**
    return INVALID
  if $e \notin \left[2^{\ell_e-1}, 2^{\ell_e-1}+2^{\ell'_e-1}\right]$ **then**
    return INVALID
  if $Z \not\equiv A^e \cdot S^v \cdot R_0^s \cdot \prod_{i=1}^{l} R_i^{a_i} \pmod{n}$ **then**
    return INVALID
  return VALID

5.3 Selective Disclosure

This section describes the card operations in the issuance protocol.

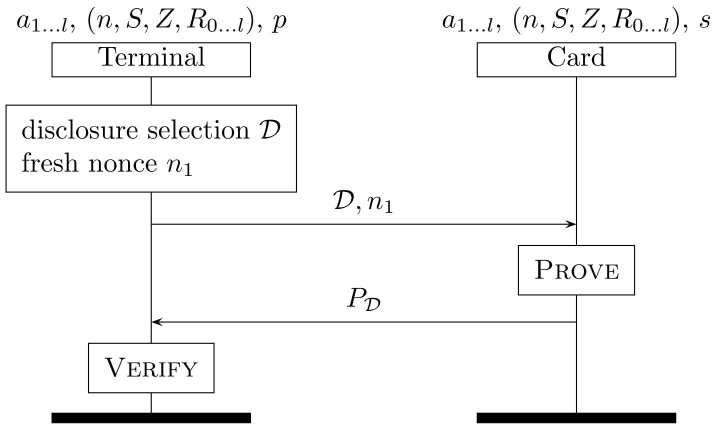

Figure 5.2: Idemix proving protocol.

5.3.1 Prove Attributes

Algorithm 12 Prove the attributes

function PROVE($\mathcal{D}, n_1$)
 $\tilde{e} \in_R \{0,1\}^{\ell_{e'}+\ell_\varnothing+\ell_H}$
 $\tilde{v}' \in_R \{0,1\}^{\ell_v+\ell_\varnothing+\ell_H}$
 $r_A \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$
 for each $i \notin \mathcal{D}$ **do**
  $\tilde{a}_i \in_R \{0,1\}^{\ell_m+\ell_\varnothing+\ell_H}$
 $A' \leftarrow A \cdot S^{r_A} \mod n$
 $\tilde{Z} \leftarrow A'^{\tilde{e}} \cdot S^{\tilde{v}'} \cdot \prod_{i \notin \mathcal{D}} R_i^{\tilde{a}_i} \mod n$
 $c \leftarrow \mathcal{H}(\texttt{context}||A'||\tilde{Z}||n_1)$
 $e' \leftarrow e - 2^{\ell_e - 1}$ ▷ Note: no modular reduction, $e' \in [0, 2^{\ell_{e'}-1}]$.
 $v' \leftarrow v - e \cdot r_A$ ▷ Note: no modular reduction.
 $\hat{e} \leftarrow \tilde{e} + c \cdot e'$ ▷ Note: no modular reduction.
 $\hat{v}' \leftarrow \tilde{v}' + c \cdot v'$ ▷ Note: no modular reduction.
 for each $i \notin \mathcal{D}$ **do**
  $\hat{a}_i = \tilde{a}_i + c \cdot a_i$ ▷ Note: no modular reduction.
 return $P_{\mathcal{D}} = (c, A', \hat{e}, \hat{v}', \{\hat{a}_i\}_{i \notin D}, \{a_i\}_{i \in D})$

Algorithm 13 Verify the attributes

function VERIFY($(c, A', \hat{e}, \hat{v}', \{\hat{a}_i\}_{i \notin D}, \{a_i\}_{i \in D}), n_1$)
 if $\hat{a}_{i \notin \mathcal{D}} \notin \pm\{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+1}$ **then**
  return INVALID
 if $\hat{e} \notin \pm\{0,1\}^{\ell_{e'}+\ell_\varnothing+\ell_H+1}$ **then**
  return INVALID
 $\hat{Z} \leftarrow \left(\frac{Z}{(\prod_{i \in \mathcal{D}} R_i^{a_i}) \cdot (A')^{2^{\ell_e - 1}}}\right)^{-c} \cdot (A')^{\hat{e}} \cdot \left(\prod_{i \notin \mathcal{D}} R_i^{\hat{a}_i}\right) \cdot S^{\hat{v}'} \mod n$
 $\hat{c} \leftarrow \mathcal{H}(\texttt{context}||A'||\hat{Z}||n_1)$
 if $c \neq \hat{c}$ **then**
  return INVALID
 return VALID