Collusion-Resistant Domain-Specific Pseudonymous Signatures[⋆⋆⋆]

Julien Bringer[1], Hervé Chabanne[1,2], Alain Patey[1,2]

[1] Morpho
surname.name@morpho.com
[2] Télécom ParisTech
surname.name@telecom-paristech.fr
Identity and Security Alliance (The Morpho and Télécom ParisTech Research Center)

Abstract. At ISC 2012, Bender *et al.* introduced the notion of domain-specific pseudonymous signatures for ID documents. With this primitive, a user can sign with domain-specific pseudonyms, that cannot be linked across domains but that are linkable in a given domain. However, their security model assumes non-collusion of malicious users, which is a strong assumption. We therefore propose improvements to their construction. Our main contribution is a new pseudonymous signature scheme based on group signatures that is collusion-resistant.

1 Introduction

Several security mechanisms [1] are sequentially used when a *machine readable travel document* (MRTD) connects, via a reader, to a service provider:

- Using *Password Authenticated Connection Establishment* (PACE), the MRTD and the reader establish a secure channel, once the MRTD user has entered his password;
- Using *Extended Acces Control* (EAC), the MRTD and the service provider authenticate to each other and establish another secure channel;
- Optionally, using *Restricted Identification* (RI), the MRTD derives a pseudonym for the service such that the service provider can link the sessions when this particular user accessed the service, but such that it is impossible for two providers to link interactions of one user in their respective domains.

The original Restricted Identification of [1] is close to a Diffie-Hellman key exchange, with static keys, as can be seen in Figure 2. The authenticity of the pseudonym sent by the card is not guaranteed. Furthermore, the domain-specific pseudonym of a user has limited applications, which motivates the work of Bender *et al.* [3], who suggest to use this pseudonym for digital signatures.

⋆ This work has been partially funded by the European FP7 FIDELITY project (SEC-2011-284862). The opinions expressed in this document only represent the authors' view. They reflect neither the view of the European Commission nor the view of their employer.

⋆⋆ This paper is an extended version of the eponymous article published at NSS 2013 [6].

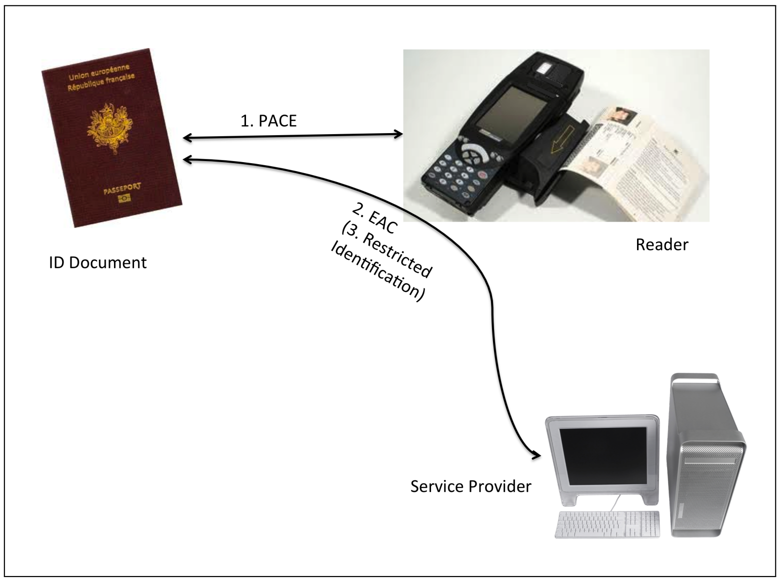

Fig. 1. Advanced Security Mechanisms for MRTDs

Inputs:

– User U: Secret key x
– Service Provider SP: Domain Identifier R

Protocol:

1. SP sends R to U.
2. U computes $nym = Hash(R^x)$, the domain-specific pseudonym of U for SP.
3. U sends nym to SP.
4. SP optionally checks if nym belongs to a black list and/or a white list.

Fig. 2. The Original Restricted Identification Protocol

To augment security guarantees and provide the possibility of using the domain-specific pseudonyms as signature keys, Bender *et al.* [3] introduced the notion of *domain-specific pseudonymous signatures*. This can be seen as a relaxation of the notion of group signatures [9, 2]. Group signatures enable users to anonymously sign on behalf of a group. Anonymity guarantees are very strong: two signatures of the same user can only be linked by the group manager, who also issues keys. However, pseudonyms produced by the RI protocol for a given user and a given service should enable linkability, since pseudonyms serve as identifiers for the users.

Domain-specific pseudonymous signatures, as defined in [3], satisfy 3 properties: *unforgeability*, *cross-domain anonymity* and *seclusiveness*. The second property states that pseudonymous signatures cannot be linked across domains while seclusiveness is a relaxation of the notion of *traceability* [2] for group signatures.

Bender *et al.* [3] suggest a modification to the original Restricted Identification protocol [1] using pseudonymous signatures, as summed up in Figure 3. In addition to his domain identifier, the service provider sends a message to be signed. This message is signed by the user using a pseudonymous signature. This pseudonymous signature at the same time guarantees that the user owns a valid unrevoked signature key and that the pseudonym is legitimate, *i.e.* that the same user key has been used to sign and to derive the pseudonym.

Inputs:

- User U: Pseudonymous signature key x
- Service Provider SP: Domain Identifier R

Protocol:

1. SP sends R and a message m to U.
2. U derives a pseudonym nym from x and R.
3. U signs the message m using x and nym
4. U sends the signature σ and nym to SP.
5. SP checks σ, using public parameters and nym.

Fig. 3. The Restricted Identification Protocol with Pseudonymous Signatures

The pseudonymous signature scheme of Bender *et al.* [3] fulfills the security requirements exposed above (unforgeability, cross-domain anonymity and seclusiveness). However, the security model relies on a very strong assumption: they assume that no two malicious users can retrieve their keys and collude. Indeed, the relation between signature keys and issuing key being linear, with two signature keys, one is able to retrieve the issuing key and thus to issue as many new valid keys as they want. To justify their assumption, the authors invoke the fact that these keys are supposed to be stored on smartcards on ID documents, and that smartcards are supposed to be tamper-proof. It is however likely that, on a national scale, two users will be able to retrieve keys stored on their smartcards.

This is the motivation for our work. We propose a new construction of pseudonymous signatures, where we allow the users to access their keys and where collusion of malicious users does not break the security guarantees. To do so, we use (collusion-resistant) group signatures and combine them with Schnorr proofs of knowledge to guarantee validity of pseudonyms.

2 Group Signatures

The generic setting for dynamic group signatures is described in [2]. We here focus on the case of Verifier-Local Revocation (VLR) group signatures [5]. The definitions and properties described in this section may only apply to the VLR case.

2.1 Setting

We consider the VLR group signature model [5, 10, 8], without Backward Unlinkability. There are three types of entities in the VLR group signature model:

– a Group Manager GM;
– a set of members;
– a set of verifiers.

A VLR Group Signature Scheme with Exculpability consists of the following algorithms:

GSKeyGen(k)**:** On input a security parameter k, this algorithm, run by GM outputs the group public parameters gpk and the issuing key ik. It also sets an empty Revocation List RL. This list will be later filled with the revocation tokens of the revoked users.

GSJoin$(gpk, ik; gpk)$**:** This algorithm is an interactive protocol between GM and a member M_i. GM takes as input the public parameters gpk and the issuing key ik, M_i only takes gpk.
In the end, M_i outputs an identity id_i, a secret key sk_i, a credential cre_i and a tracing key tk_i (included in cre_i). GM gets id_i and tk_i and outputs also a revocation token rt_i for M_i.

GSRevoke(gpk, rt_i, RL)**:** GM runs this algorithm to prevent a member M_i from making valid signatures. It outputs an updated revocation list RL, where rt_i has been added.

GSSign(gpk, sk_i, cre_i, m)**:** This algorithm, run by a member M_i, takes as input a message m, M_i's keys sk_i and cre_i and a message m to sign. It outputs a signature σ.

GSVerify(gpk, RL, m, σ)**:** This algorithm, run by a verifier takes as input a message m, its signature σ, the Revocation List RL and the public parameters gpk. It checks if the message has been signed by an unrevoked group member, without revealing the signer's identity. The possible outputs are **valid** and **invalid**.

GSOpen$(gpk, m, \sigma, \{tk_i\}_i)$**:** This algorithm is run by GM. It takes a signature σ on a message m as input, together with all tracing keys of the group. It reveals the tracing key tk_i and the identity id_i of the signer. If it fails to recover the identity of the signer, it returns $\perp$.

2.2 Security Requirements

We require our scheme to satisfy *Correctness*, *Selfless-Anonymity*, *Traceability* and *Exculpability*. Let us first define *Correctness*.

Correctness The scheme is *correct* if every signature created by an unrevoked member is verified as valid.

The other properties are game-based. These games are summed up below.

Selfless-Anonymity Selfless-Anonymity guarantees that no one but the (unrevoked) signer and the GM can gain information about the signer of a given signature.

Consider the following Selfless-Anonymity game played by an adversary $\mathcal{A}$:

Setup: The challenger $\mathcal{C}$ runs *GSKeyGen*(k) and obtains *gpk* and *ik*. He sends *gpk* to $\mathcal{A}$.

Queries: $\mathcal{A}$ can make *Join*, *Sign*, *Corruption*, *Revocation* and *Open* queries to the challenger $\mathcal{C}$.

Challenge: Adversary $\mathcal{A}$ outputs a message m^* and two members M_{i_0} and M_{i_1}, who are neither corrupted, nor revoked. $\mathcal{C}$ chooses $b \in_R \{0, 1\}$ and sends $\sigma^* = GSSign(gpk, sk_{i_b}, cre_{i_b}, m^*)$ to $\mathcal{A}$.

Restricted Queries: Adversary $\mathcal{A}$ can make the same queries as in the *Queries* phase, as long as he does not require the revocation or the corruption of M_{i_0} or M_{i_1}, nor the opening of σ^*.

Output: Adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ on b.

The scheme is said to satisfy the *Selfless-Anonymity* property if the probability $|\Pr[b = b'] - 1/2|$ is negligible for any probabilistic polynomial-time adversary $\mathcal{A}$.

Traceability The scheme is *traceable* if an adversary $\mathcal{A}$ is unable to forge a valid signature that cannot be opened properly. Consider the following *Traceability* game played by $\mathcal{A}$:

Setup: The challenger $\mathcal{C}$ runs *KeyGen*(k) and obtains *gpk* and *ik*. He sends *gpk* to $\mathcal{A}$. He also sets an empty revocation list *RL*.

Queries: Adversary $\mathcal{A}$ can execute *Join*, *Corrupt*, *Sign* and *Open* queries.

Output: Adversary $\mathcal{A}$ outputs a message m^* and a signature σ^*.

Adversary $\mathcal{A}$ wins the game if:

1. $GSVerify(gpk, RL, m^*, \sigma^*) = valid$ (implying that σ^*'s opening traces to a member outside the coalition);
2. Adversary $\mathcal{A}$ did not obtain σ^* by making a Sign query on m^*.

The scheme satisfies *Traceability* if no polynomial probabilistic adversary is able to win the above game with a non-negligible probability.

Exculpability The aim of the *Exculpability* property is to offer protection against the Group Manager. In the *Exculpability* game, roles are inverted: the adversary is the GM and, consequently, knows the group's secret key and all the players' credentials. The goal of the adversary is to forge a valid signature that will be attributed to an honest (*i.e.* not corrupted) member by the *GSOpen* algorithm. This signature must be such that it cannot be denied by the signer. Consider the following *Exculpability* game played by an adversary $\mathcal{A}$:

Setup: Challenger $\mathcal{C}$ runs $GSKeyGen(k)$ and obtains gpk and ik. $\mathcal{C}$ stores gpk and sends gpk and ik to $\mathcal{A}$[3].

Queries: Adversary $\mathcal{A}$ can request *Join*, *Corrupt* and *Sign* queries.

Output: Adversary $\mathcal{A}$ outputs a message m^*, the Revocation List RL, a signature σ^* and a tracing key pair (id_{i^*}, tk_{i^*}).

Adversary $\mathcal{A}$ wins the game if all the following statements hold:

1. He did not obtain σ^* from making a query on m^*;
2. $GSVerify(gpk, RL, m^*, \sigma^*) = \textbf{valid}$;
3. $GSOpen(gpk, m^*, \sigma^*, \{tk_i\}) = (id_{i^*}, tk_{i^*})$;
4. He did not corrupt M_{i^*};
5. It is impossible for M_{i^*} to prove the knowledge of a member key $(id_i$, sk_i, $cre_i)$ such that $sk_i \neq sk_{i^*}$ and such that sk_i could have issued σ^* as a valid signature[4].

Note that the last requirement is to ensure protection against a dishonest opening from the manager. The scheme satisfies *Exculpability* if no polynomial probabilistic adversary is able to win the above game with a non-negligible probability.

2.3 The CL-BP Scheme

We here detail the scheme of Chen and Li [10], patched by Bringer and Patey [8] to ensure Traceability, that we denote by CL-BP. This scheme is well-suited for our purpose, since one can efficiently prove that the same key has been used to generate a pseudonym and to sign the message. Furthermore, the CL-BP scheme enjoys efficient revocation checks.

The *GSKeyGen* algorithm is described in Algorithm 1. The issuing key is $\gamma \in_R \mathbb{Z}_p$, its public counterpart is $w = g_2^{\gamma}$.

Algorithm 1 $GSKeyGen(k)$

```
1: Choose bilinear groups $G_1, G_2, G_\mathcal{T}$ of order a $k$-bit prime number $p$ that is safe (i.e. $(p-1)/2$ prime number), a prime number $q$ and a pairing $e : G_1 \times G_2 \to G_\mathcal{T}$. Let $g_1, g_2$ be generators of $G_1$ and $G_2$.
2: Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$
3: Choose $\tilde{g}_1, \hat{g}_1 \in_R G_1, \gamma \in_R \mathbb{Z}_p^*$ and compute $w = g_2^{\gamma}$.
4: Compute $T_1 = e(g_1, g_2)$, $T_2 = e(\tilde{g}_1, g_2)$, $T_3 = e(\hat{g}_1, g_2)$ and $T_4 = e(\hat{g}_1, w)$.
5: Output: $gpk = (G_1, G_2, G_\mathcal{T}, e, p, g_1, g_2, \tilde{g}_1, \hat{g}_1, w, H, T_1, T_2, T_3, T_4)$ and $ik = \gamma$.
```

The *GSJoin* algorithm is explained in Algorithm 2. Each member M_i chooses a secret key $sk_i = f_i \in_R \mathbb{Z}_p$, not known by GM. M_i gives to GM an identity $id_i = \tilde{g}_1^{f_i}$ and proves the knowledge of f_i. GM sends him, over a secure channel, a credential $cre_i = (A_i, x_i)$, that satisfies $e(A_i, wg_2^{x_i}) = e(g_1\tilde{g}_1^{f_i}, g_2)$. The revocation token for a member M_i is $rt_i = x_i$ and $tk_i = x_i$ is the tracing key.

[3] This corresponds to a trusted setup run on behalf of the adversary.

[4] This condition is formalized in [10] by using two functionalities called DProve and DVerify.

Algorithm 2 *GSJoin*$(gpk,\ ik\ ;\ gpk)$

```
1: GM sends a nonce $n_i \in \{0,1\}^k$ to $M_i$.
2: $M_i$ chooses $f_i \in_R \mathbb{Z}_p$ and computes $F_i = \tilde{g}_1^{f_i}$. He sets $sk_i = f_i$ and $id_i = F_i$. He
   chooses $r_f \in_R \mathbb{Z}_p$ and computes $R = \tilde{g}_1^{r_f}$. He computes $c = H(gpk||F_i||R||n_i)$ then
   $s_f = r_f + cf_i$.
3: $M_i$ sends $comm = (F_i, c, s_f)$ to GM.
4: GM computes $R' = \tilde{g}_1^{s_f} F_i^{-c}$ and checks that $c = H(gpk||F||R'||n_i)$ and $s_f \in \mathbb{Z}_p$.
   He chooses $x_i \in_R \mathbb{Z}_p$ and computes $A_i = (g_1 F_i)^{1/(x_i+\gamma)}$. He sets $cre_i = (A_i, x_i)$,
   $tk_i = x_i$ and $id_i = F_i$.
5: GM sends $cre_i$ to $M_i$, using a secure channel.
6: $M_i$ checks that $e(A_i, wg_2^{x_i}) = e(g_1 \tilde{g}_1^{f_i}, g_2)$ and outputs $(id_i, sk_i, cre_i)$.
7: The revocation token for $M_i$ is $rt_i = x_i$.
```

The *GSSign* algorithm is described in Algorithm 3. When a member M_i creates a signature, he first chooses a random $B \in_R G_1$ and computes $J = B^{f_i}$, $K = B^{x_i}$. He picks a random $a \in_R \mathbb{Z}_p$, computes $b = ax_i$ and $T = A_i \hat{g}_1^a$. He then does a NIZK PK of (f_i, A_i, x_i) satisfying $J = B^{f_i}$, $K = B^{x_i}$ and $e(A_i, wg_2^{x_i}) = e(g_1 \tilde{g}_1^{f_i}, g_2)$.

Algorithm 3 *GSSign*(gpk, sk_i, cre_i, m)

```
1: Choose $B \in_R G_1$ and compute $J = B^{f_i}$, $K = B^{x_i}$
2: Choose $a \in_R \mathbb{Z}_p$, compute $b = ax_i$ and $T = A_i \hat{g}_1^a$.
3: Choose $r_f, r_x, r_a, r_b \in_R \mathbb{Z}_p$.
4: Compute $R_1 = B^{r_f}$, $R_2 = B^{r_x}$, $R_4 = K^{r_a} B^{-r_b}$, $R_3 = e(T, g_2)^{-r_x} T_2^{r_f} T_3^{r_b} T_4^{r_a}$.
5: Compute $c{=}H(gpk||B||J||K||T||R_1||R_2||R_3||R_4||m)$.
6: Compute $s_f = r_f + cf_i$, $s_x = r_x + cx_i$, $s_a = r_a + ca$ and $s_b = r_b + cb$.
7: Output: $\sigma = (B, J, K, T, c, s_f, s_x, s_a, s_b)$.
```

The *GSVerify* algorithm is described in Algorithm 4. To check a signature, the verifier checks the proof of knowledge. If the check succeeds, he also does a Revocation Check: $\forall rt_{i'} = x_i' \in RL$, he checks that $K \neq B^{rt_{i'}}$.

Algorithm 4 *GSVerify*(gpk, m, σ, RL)

```
1: **Signature Check:**
2:     Check that $B, J, K, T \in G_1$ and $s_f,\ s_x,\ s_a,\ s_b \in \mathbb{Z}_p$.
3:     Compute $R_1' = B^{s_f} J^{-c}$, $R_2' = B^{s_x} K^{-c}$, $R_4' = K^{s_a} B^{-s_b}$ and $R_3' =$
   $e(T, g_2)^{-s_x} T_2^{s_f} T_3^{s_b} T_4^{s_a} T_1^c e(T, w)^{-c}$ .
4:     Check that $c{=}H(gpk||B||J||K||T||R_1'||R_2'||R_3'||R_4'||m)$.
5: **Revocation Check:**
6:     Check that $\forall rt_i \in RL, K \neq B^{rt_i}$.
7: Output **valid** if all checks succeed. Otherwise output **invalid**.
```

The *GSOpen* algorithm consists in an application of the *Revocation Check* part of the *GSVerify* algorithm, using the revocation tokens of all enrolled users. When there is an equality between K and B^{rt_i} (see Algorithm 4, line 6), the corresponding user is output.

The security of the CL-BP scheme is proved in [8]. We sum up the security guarantees in the following theorem, where DDH stands for Decisional Diffie-Hellman, q-SDH for q-Strong Diffie Hellman [4] and DLP for Discrete Logarithm Problem.

Theorem 1 (Security of the CL-BP Scheme). *The CL-BP scheme achieves*

- *Correctness;*
- *Selfless-Anonymity, under the DDH assumption;*
- *Traceability, under the q-SDH assumption;*
- *Exculpability, under the DLP assumption.*

3 Pseudonymous Signatures

We give in this section definitions that follow our stronger security model and indicate where the weaker security model of [3] differs.

3.1 Setting

Three entities are involved:

- The Issuing Authority (IA)
- Users M_i
- Service Providers SP_j with specific domains D_j

The issuing authority issues secret keys to the users and domain parameters to the service providers. The users can generate, from their secret keys and the domain parameters, one pseudonym per domain. Using domain-specific pseudonymous signatures, users can sign messages linked to their pseudonyms. The pseudonym being attached to the signatures, the signatures of one user in one domain are obviously linkable. The construction guarantees that signatures of the same user for different domains are not linkable by (even colluding) service providers. However, the issuing authority can retrieve these links.

There are six algorithms in the scheme:

PSKeyGen(k) The Issuing Authority generates a master secret key msk and a master public key gpk.
The IA and the SP_j's generate domain-specific public keys dpk_j.

PSJoin(msk, gpk) The Issuing Authority and the user M_i interact to generate a secret key sk_i for user M_i.

PSSign(gpk,sk_i,dpk_j,m) The user M_i outputs his pseudonym nym_{ij} for domain D_j and a signature σ on m for domain D_j

PSVerify(gpk,nym_{ij},dpk_j,m,σ,RL_j) The Service Provider for domain D_j checks the signature σ on message m and the link between σ and nym_{ij}. SP_j also performs a revocation check, using RL_j, the list of revoked pseudonyms for domain D_j

PSDomainRevoke(RL_j,nym_{ij}) SP_j runs this algorithm to prevent a member M_i from making valid signatures in D_j. It outputs an updated revocation list RL_j, where nym_{ij} has been added.

PSRevoke(gpk,x_i) IA runs this algorithm to prevent a member M_i from making valid signatures in all domains. He sends an information x_i about the key of M_i that the SP_j's use to revoke M_i from their domains. Every service provider SP_j outputs an updated RL_j.

3.2 Security

The security properties of pseudonymous signatures are close to the properties of group signatures. We require a pseudonymous signature scheme to satisfy *Correctness*, *Cross-Domain Anonymity*, *Unforgeability* and *Seclusiveness.*

Correctness The scheme is *correct* if every signature-pseudonym couple created by an unrevoked member is verified as valid.

Cross-Domain Anonymity Cross-Domain Anonymity guarantees that signatures are anonymous and that linkability is possible within a given domain only, even with colluding service providers.
Consider the following Cross-Domain Anonymity game played by an adversary $\mathcal{A}$:

Setup: The challenger $\mathcal{C}$ runs $KeyGen(k)$ and obtains gpk and msk. He sends gpk to $\mathcal{A}$.

Queries: $\mathcal{A}$ can make *Join*, *Sign*, *User Corruption*[5], *Service Provider Corruption*, *Domain-Specific Revocation* and *Revocation* queries to the challenger $\mathcal{C}$.

Challenge: Adversary $\mathcal{A}$ outputs a message m^*, a domain D_j and two members M_{i_0} and M_{i_1}, who are neither corrupted, nor revoked for domain D_j. Furthermore, no *Sign* request in the domain D_j for these two members must have been called. $\mathcal{C}$ chooses $b \in_R \{0, 1\}$ and sends $\sigma^* = PSSign(gpk, sk_{i_b}, dpk_j, m^*)$ to $\mathcal{A}$.

Restricted Queries: Adversary $\mathcal{A}$ can make the same queries as in the *Queries* phase, as long as he does not require the revocation or the corruption of M_{i_0} or M_{i_1}, nor signatures of M_{i_0} or M_{i_1} for domain D_j.

Output: Adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ on b.

The scheme is said to satisfy the *Cross-Domain Anonymity* property if the probability $|\Pr[b = b'] - 1/2|$ is negligible for any probabilistic polynomial-time adversary $\mathcal{A}$.

Seclusiveness The scheme achieves *Seclusiveness* if an adversary $\mathcal{A}$ is unable to forge a valid signature that cannot be opened properly. Consider the following *Seclusiveness* game played by $\mathcal{A}$:

[5] In the security definition of the scheme of [3], only one corruption is authorized

Setup: The challenger $\mathcal{C}$ runs $KeyGen(k)$ and obtains gpk and ik. He sends gpk to $\mathcal{A}$. He also sets empty revocation list RL_j.

Queries: Adversary $\mathcal{A}$ can execute *Join*, *Corrupt*[6], *Sign* and *Open* queries.

Output: Adversary $\mathcal{A}$ outputs a message m^* and a signature σ^*.

Adversary $\mathcal{A}$ wins the game if:

1. $PSVerify(gpk, nym_{ij}, dpk_j, RL_j, m^*, \sigma^*) = valid$ (implying that σ^*'s opening traces to a member outside the coalition);
2. Adversary $\mathcal{A}$ did not obtain σ^* by making a Sign query on m^*.

The scheme satisfies *Traceability* if no polynomial probabilistic adversary is able to win the above game with a non-negligible probability.

Unforgeability The aim of the *Unforgeability* property is to prevent anyone, including the Group Manager, from making signatures on behalf of a given user. Notice that, contrary to the security definitions of [3], we do not require the group manager to delete information about the users' keys after the *PSJoin* algorithm.

In the *Unforgeability* game, roles are inverted: the adversary is the IA and, consequently, knows the group's secret key and all the players' and providers' credentials. The goal of the adversary is to forge a valid signature that will be attributed to an honest (*i.e.* not corrupted) member. This signature must be such that it cannot be denied by the signer. Consider the following *Unforgeability* game played by an adversary $\mathcal{A}$:

Setup: Challenger $\mathcal{C}$ runs $KeyGen(k)$ and obtains gpk and ik. $\mathcal{C}$ stores gpk and sends gpk and ik to $\mathcal{A}$.

Queries: Adversary $\mathcal{A}$ can request *Join*, *Corrupt* and *Sign* queries.

Output: Adversary $\mathcal{A}$ outputs a message m^*, a domain D_j, the Revocation List RL_j, a signature σ^*, a pseudonym nym_{ij} and a member M_{i^*}.

Adversary $\mathcal{A}$ wins the game if all the following statements hold:

1. He did not obtain σ^* from making a query on m^*;
2. $PSVerify(gpk, nym_{ij}, dpk_j, RL_j, m^*, \sigma^*) = \mathbf{valid}$;
3. The signature opens to M_{i^*}, and M_{i^*} cannot deny it.
4. He did not corrupt M_{i^*};

The scheme satisfies *Unforgeability* if no polynomial probabilistic adversary is able to win the above game with a non-negligible probability.

3.3 Overview of the scheme of Bender *et al.*

We sum up the components of the pseudonymous signature scheme of [3] and the reasons why it does not resist to collusions.

Let $G = \langle g \rangle$ be a cyclic group of prime order q. The secret key of the IA is made of two randomly chosen integers $x, z \in_R \mathbb{Z}_q$. The public parameters of the system are g, g^x and g^z.

The key of a user M_i is a couple (x_{1i}, x_{2i}), chosen by the IA, such that $x_{1i} = x - z \cdot x_{2i}$.

[6] In [3], only one *Corrupt* query is allowed.

The domain parameters for every domain D_j are chosen by the IA, by picking a random $r_j \in_R \mathbb{Z}_q^*$ and setting $dpk_j = g^{r_j}$. SP_j learns dpk_j, but not r_j.

The pseudonym of a user M_i for domain D_j is $dpk_j^{x_{1i}}$.

To sign, user M_i uses two intertwined proofs of knowledge, a Schnorr proof of knowledge [13] that he knows the discrete logarithm of the pseudonym and an Okamoto proof [12] that he owns a valid (x_1, x_2) key pair. The proofs are non-interactive and rely on the random oracle. The message to be signed is included in the entries of the hash function used to generate the challenge.

Limitations First, it is noticeable, that when one finds two (x_1, x_2) couples, one easily retrieves the secret keys x and z of the IA and can thus generate as many valid key couples as one wants. The authors of [3] justify that, since these keys are supposed to be stored on smartcards, assumed to be tamper-proof, security will be guaranteed. We estimate that this assumption is optimistic, especially for the sensitive application of ID documents.

Second, the IA generates all the keys and, thus, is able to sign on behalf of the users. It is specified in [3] that these keys are supposed to be deleted by the IA, once delivered to the users, which is impossible to verify. This might raise some security issues, we also solve this problem in our proposal thanks to the *Exculpability* property of group signatures.

4 Our Proposal for a Collusion-Resistant Pseudonymous Signature Scheme

4.1 The Proposal

One of the main properties of group signatures is that they prevent from linking signatures. However, pseudonymous signatures are supposed to enable linkability of signatures if they are accompanied by the same pseudonym, but unlinkability otherwise. By combining the construction of [3] with the CL-BP group signature, we build pseudonymous signatures that are resistant to collusions. This construction requires more computation from the user to sign messages. It requires elliptic curve cryptography but the smartcard does not need to be able to compute pairings.

The *PSKeyGen* algorithm, described in Algorithm 5 is mostly an execution of the *GSKeyGen algorithm* of the CL-BP scheme by the Issuing Authority. IA also has to pick randomly chosen domain parameters $dpk_j = g_1^{r_j}$ that he sends to the service providers. It is essential, in order to preserve unlinkability, that the service providers do not learn r_j.

The *PSJoin* algorithm, described in Algorithm 6[7] consists in an execution of the *GSJoin* algorithm.

[7] Optionally, during the execution of this algorithm, M_i also gets $e(A_i, g_2)$ to avoid pairing computations by the smartcard

Algorithm 5 *PSKeyGen(k)*

```
1: Run the GSKeyGen algorithm of the CL-BP scheme, as Group Manager.
2: Thus obtain $gpk$ and $msk = \gamma$
3: For every service provider $SP_j$, pick a random $r_j \in_R \mathbb{Z}_p$ and issue the domain
   parameters $dpk_j = g_1^{r_j}$ to $SP_j$.
```

Algorithm 6 *PSJoin(msk, gpk)*

```
1: IA runs a GSJoin algorithm with the user $M_i$.
2: $M_i$ gets a key $gsk_i = (x_i, A_i, f_i)$
3: IA gets $x_i$ and $A_i$.
```

The *PSSign* algorithm, described in Algorithm 7 consists in a CL-BP signature, a domain-specific pseudonym derivation, and a proof of knowledge that the same key has been used in both operations.

Algorithm 7 *PSSign(gpk,sk_i,dpk_j,m)*

```
1: Choose $B \in_R G_1$ and compute $J = B^{f_i}$, $K = B^{x_i}$.
2: Choose $a \in_R \mathbb{Z}_p$, compute $b = ax_i$ and $T = A_i \hat{g}_1^a$.
3: Choose $r_f, r_x, r_a, r_b \in_R \mathbb{Z}_p$.
4: Compute $R_1 = B^{r_f}$, $R_2 = B^{r_x}$, $R_4 = K^{r_a} B^{-r_b}$, $R_3 = e(T, g_2)^{-r_x} T_2^{r_f} T_3^{r_b} T_4^{r_a}$ and
   $R_5 = dpk_j^{r_x}$.
5: Compute $c$=$H(gpk||B||J||K||T||R_1||R_2||R_3||R_4||R_5||m)$.
6: Compute $s_f = r_f + cf_i$, $s_x = r_x + cx_i$, $s_a = r_a + ca$ and $s_b = r_b + cb$.
7: Output: $\sigma = (B, J, K, T, c, s_f, s_x, s_a, s_b)$ and $nym_{ij} = dpk_j^{x_i}$
```

The *PSVerify* algorithm, described in Algorithm 8 is a verification of the proof of knowledge, followed by a revocation check, where the verifier checks if the pseudonym is on the revocation list or not. Notice that the verifier only performs a list membership test and not a linear number of arithmetic operations, as in the Revocation Check of the *GSVerify* algorithm.

Algorithm 8 *PSVerify(gpk,nym_{ij},dpk_j,m,σ,RL_j)*

```
1: Signature Check:
2:     Check that $B, J, K, T \in G_1$ and $s_f$, $s_x$, $s_a$, $s_b \in \mathbb{Z}_p$.
3:     Compute $R'_2 = B^{s_x} K^{-c}$, $R'_3 = e(T, g_2)^{-s_x} T_2^{s_f} T_3^{s_b} T_4^{s_a} T_1^c e(T, w)^{-c}$, $R'_4 =$
   $K^{s_a} B^{-s_b}$, $R'_1 = B^{s_f} J^{-c}$ and $R'_5 = dpk_j^{s_x} nym^{-c}$.
4:     Check that $c$=$H(gpk||B||J||K||T||R'_1||R'_2||R'_3||R'_4||R'_5||m)$.
5: Revocation Check:
6:     Check that $nym_{ij} \notin RL_j$.
7: Output valid if all checks succeed. Otherwise output invalid.
```

The revocation algorithms, *PSDomainRevoke*, described in Algorithm 9, and *PSRevoke*, described in Algorithm 10, consist in adding the pseudonym of the revoked user in the revocation list RL_j of the domain RL_j.

Algorithm 9 *PSDomainRevoke(RL_j,nym_{ij})*

```
1: Add $nym_{ij}$ to $RL_j$
2: Output $RL_j$
```

Algorithm 10 *PSRevoke(gpk,x_i)*

```
1: IA sends $rt_i = x_i$ to all $SP_j's$
2: Every $SP_j$ adds $nym_{ij} = dpk_j^{rt_i}$ to $RL_j$ and outputs $RL_j$.
```

4.2 Security

Our pseudonymous signature scheme achieves the security properties described in Section 3.2, under the same security conditions as the CL-BP scheme. It can be proven in a similar way as in [10, 8], *i.e.* by showing how a successful adversary against our security games can be used to break the security assumptions. Security is guaranteed is in the random oracle model.

Theorem 2 (Correctness). *The pseudonymous signature scheme described in Section 4.1 achieves Correctness.*

Theorem 3 (Cross-Domain Anonymity). *Under the Decisional Diffie-Hellman assumption, the pseudonymous signature scheme described in Section 4.1 achieves Cross-Domain Anonymity.*

Theorem 4 (Seclusiveness). *Under the q-Strong Diffie-Hellman assumption [4], the pseudonymous signature scheme described in Section 4.1 achieves Seclusiveness.*

Theorem 5 (Unforgeability). *Under the Discrete Logarithm assumption, the pseudonymous signature scheme described in Section 4.1 achieves Unforgeability.*

Implementation In [8], parameters for the CL-BP scheme are suggested and a computation time analysis is performed. The computation required by our *PSSign* algorithm is the cost of the *GSSign* algorithm of the CL-BP scheme and an exponentiations necessary to compute R_5 and nym_{ij}. As noticed in [8], all the computation of *GSSign* but the hash function can be computed offline, before the knowledge of the message to be signed. This offline work consists in 6 multi-exponentiations in G_1 and 1 multi-exponentiation in $G_\mathcal{T}$. The remaining online work for our pseudonymous signature is then 1 hash function computation and 2 exponentiations in G_1. With the parameters of [8], this online computation requires less than 100 ms on a personal computer.

The corresponding computation for *PSVerify* is 5 multi-exponentiations in G_1, 1 multi-exponentiation in $G_\mathcal{T}$ and 1 pairing. Contrary to the one of *GSVerify*, the cost of the *Revocation Check* of *PSVerify* is negligible. With the parameters of [8], the computation of *PSVerify* requires around 500 ms on a personal computer.

5 Conclusion

We succeed in building a pseudonymous signature scheme that resists to collusions, which improves on the previous proposal of Bender *et al.* [3]. This property is an asset for a secure large-scale deployment of this mechanism on electronic ID documents. The price to pay is the use of elliptic-curve cryptography and pairings. However, it can be deployed such that no pairings are computed on the users' smartcards.

Our proposal relying on VLR group signatures, it can be extended to enjoy additional features that have already been proposed for group signatures such as biometric key derivation [7] or Backward Unlinkability [11, 8]. The latter property enables to have a time period-specific revocation process.

References

1. Advanced security mechanisms for machine readable travel documents. part 2 extended access control version 2 (EACv2), password authenticated connection establishment (PACE), and restricted identification (RI). Tech. Rep. TR-03110-2, BSI (March 2012), version 2.10
2. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. IACR Cryptology ePrint Archive 2004, 77 (2004)
3. Bender, J., Dagdelen, Ö., Fischlin, M., Kügler, D.: Domain-specific pseudonymous signatures for the german identity card. In: Gollmann, D., Freiling, F.C. (eds.) ISC. Lecture Notes in Computer Science, vol. 7483, pp. 104–119. Springer (2012)
4. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3027, pp. 56–73. Springer (2004)
5. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security. pp. 168–177. ACM (2004)
6. Bringer, J., Chabanne, H., Patey, A.: Collusion-resistant domain-specific pseudonymous signatures. In: International Conference on Network and System Security (NSS) (2013)
7. Bringer, J., Chabanne, H., Pointcheval, D., Zimmer, S.: An application of the Boneh and Shacham group signature scheme to biometric authentication. In: Matsuura, K., Fujisaki, E. (eds.) IWSEC. Lecture Notes in Computer Science, vol. 5312, pp. 219–230. Springer (2008)

8. Bringer, J., Patey, A.: VLR group signatures - how to achieve both backward unlinkability and efficient revocation checks. In: SECRYPT. pp. 215–220 (2012)
9. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 547, pp. 257–265. Springer (1991)
10. Chen, L., Li, J.: VLR group signatures with indisputable exculpability and efficient revocation. In: Elmagarmid, A.K., Agrawal, D. (eds.) SocialCom/PASSAT. pp. 727–734. IEEE Computer Society (2010)
11. Nakanishi, T., Funabiki, N.: A short verifier-local revocation group signature scheme with backward unlinkability. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., ichi Kawamura, S. (eds.) IWSEC. Lecture Notes in Computer Science, vol. 4266, pp. 17–32. Springer (2006)
12. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 740, pp. 31–53. Springer (1992)
13. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptology 4(3), 161–174 (1991)