Domain-Specific Pseudonymous Signatures Revisited

Kamil Kluczniak

Wrocław University of Technology, Department of Computer Science
kamil.kluczniak@pwr.edu.pl

Abstract. Domain-Specific Pseudonymous Signature schemes were recently proposed for privacy preserving authentication of digital identity documents by the BSI, German Federal Office for Information Security. The crucial property of domain-specific pseudonymous signatures is that a signer may derive unique pseudonyms within a so called domain. Now, the signer's true identity is hidden behind his domain pseudonyms and this pseudonyms are unlinkable, i.e. it is infeasible to correlate two pseudonyms with a single user. In this paper we take a critical look at the security definitions and constructions of domain-specific pseudonymous signatures proposed by far. We review two articles which propose "sound and clean" security definitions and point out some issues present in this models. Some of this issues may have a strong practical impact on constructions provable secure in this models. Additionally, we point out some worrisome facts about the proposed schemes and their security analysis.

Key words: eID Documents, Privacy, Domain Signatures, Pseudonymity, Security Definition, Provable Security

1 Introduction

Domain signature schemes are signature schemes where we have a set of users, an issuer and a set of domains. Each user obtains his secret keys in collaboration with the issuer and then may sign data with regards to his pseudonym. The crucial property of domain signatures is that each user may derive a pseudonym within a domain. Domain pseudonyms of a user are constant within a domain and a user should be unable to change his pseudonym within a domain, however, he may derive unique pseudonyms in each domain of the system. Moreover, a domain owner (verifier) should be convinced that the signer was admitted by the issuer. In short domain signature schemes need to fulfil the following properties:

– Unforgeability - it is infeasible for a coalition of malicious users and domain holders to produce a signature on behalf of a honest user.
– Seclusiveness - it is infeasible for a group of malicious users (possibly all users in the system) to create a signature on behalf of a user which was not admitted by the issuer. In other words, it is infeasible for any coalition to produce false identities.

– Unlinkability - having two or more pseudonyms from different domains it is infeasible to tell whether this pseudonyms came from a single user or different users.

The concept of Domain Signatures was first introduced in a short paper [1] and was inspired by the design goals for Restricted Identification protocols[1] for electronic identity cards stated by the German Federal Office for Information Security (BSI). The proof of concept construction from [1] requires a user to obtain a certificate for each pseudonym in each domain.

Later the idea was developed in [2] and then continued in [3]. The paper [2] first introduced a definition for domain signatures and proposed a construction where users may derive domain pseudonyms and produce signatures without the need to obtain a certificate for each domain separately. The authors from [3] point out some serious issues with the definition and construction proposed in [2]. They also introduced a security definition which allegedly solves the issues present in previous work and design an efficient domain signature scheme based on groups with bilinear maps.

In this article we focus on the papers [2] and [3]. We recall the issues present in [2] pointed out by [3] and give some additional comments. Then, we review the work from [3] pointing out some issues and even mistakes in the security definition, construction and security analysis. The paper [3] was also a significant contribution in the thesis [4]. In our opinion, because of the complexity of scientific writing, this issues may be overlooked by potential implementers or even specialist in the field what may be significant for the security of the end user. Significance, to our critical review gives the fact that [2] and [3], may be considered for standardization according to the technical guideline [5] and further implementation on identity documents.

Structure of the Paper. In Section 2 we recall the work from [2] and describe some issues and mistakes partially identified in [3]. In Section 3 we first recall the security definitions from [3]. Then, we discuss some controversial parts of the definitions. The crucial part is the unlinkability which allegedly captures the "dynamic case" i.e. the case where users may be admitted to the group at any point of the lifetime of the system. We will give some critical arguments on the unlinkability definition and the definition of other properties, which may result in "unnatural" (stupid) schemes but provably secure in the proposed model. In a further version of the paper we plan to show such constructions and proof that they fulfil all defined requirements. Finally, we show some issues according to the implementation of the protocol. As the construction from [3] requires time consuming operations from a signer and the prime implementation target is supposed to be a smart card, the authors propose an extension of the definition and the signature scheme to outsource come computation to a reader. We will in particular show, some overcomings present in the security models and security analysis, and discuss the practical impact of this issues.

[1] Interactive identification protocols which need to fulfil similar properties as Domain Signatures.

In our review we will recall some definitions from [2] and [3], and use the original notation and terminology from these papers. In order to make our paper as self contained as possible, we additionally recall the signature schemes from [2] and [3] in Appendix A and B. However, for detailed security analysis etc. we refer to the original papers.

2 Domain-Specific Pseudonymous Signatures for the German Identity Card

In this section we review the security models and constructions from [2]. We will point out some mistakes and flaws, some of which have been already disclosed in [3]. While the paper [2] is merely a research publication, the technical guide released by BSI [5] implicitly points to this algorithm.

Bellow we briefly recall the definition of the scheme and the notation as it has been introduced in [2]. For a more detailed description we refer to the original paper [2].

Definition 1. *A Domain-specific Pseudonymous Signature scheme is a collection of the following efficient algorithms* $\mathcal{NYMS} = (\mathsf{NymKGen}, \mathsf{NymDSgen}, \mathsf{NymSig}, \mathsf{NymVf})$ *defined as follows:*

$\mathsf{NymKGen}(1^\lambda, 1^n, 1^d)$ *is a probabilistic algorithm which, on input a security parameter* 1^λ *and parameters* 1^n, 1^d *(both "polynomial in* λ*" according to [2]) outputs a pair* $(\mathsf{gpk}, \mathsf{gmsk})$*, where* gpk *is the group public key and* gmsk *the secret key of the group manager, and outputs* n *(unique) pseudonyms* nym *with their corresponding secret keys* $\mathsf{gsk}[\mathsf{nym}]$*, and* d *domain descriptions* dpk*.*

$\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk})$ *is a deterministic algorithm which maps a pseudonym* nym *(and its secret key* $\mathsf{gsk}[\mathsf{nym}]$*) and the domain description* dpk *to a domain-specific pseudonym* $\mathsf{dsnym} = \mathsf{nym}[\mathsf{dpk}]$ *of the user with the pseudonym* nym*.*

$\mathsf{NymSig}(\mathsf{dsnym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}, m)$ *is a probabilistic algorithm which, on input a domain-specific pseudonym* dsnym*, a secret key* $\mathsf{gsk}[\mathsf{nym}]$ *of the user having domain pseudonym* dsnym*, a domain* dpk*, and message* m*, outputs a signature* σ *of* m *created with the user's private key* $\mathsf{gsk}[\mathsf{nym}]$ *and related to the domain* dpk*.*

$\mathsf{NymVf}(\mathsf{gpk}, \mathsf{dsnym}, \mathsf{dpk}, m, \sigma, B)$ *is a deterministic algorithm which, on input a message* m *and a signature* σ *together with the group public key* gpk*, a domain-specific pseudonym* dsnym*, the domain's description* dpk*, and a blacklist* B*, outputs either* 1 *(for "the signature is valid") or* 0 *(for "the signature is invalid").*

One may first observe that

– According to the definition there is a constant group of users, and a constant number of domain descriptions, where both user secret keys and domain descriptions are generated beforehand by $\mathsf{NymKGen}$.

– NymKGen also produces a secret key of the group manager, which however is not used by any algorithm in the system. Thus, we find it puzzling what is the point of returning a group manager secret key actually is? Any unused component in a system is commonly regarded as a security threat and should be removed as a potential place for installing backdoors.

Cross-Domain Anonymity. For a detailed definition of Cross-Domain Unlinkability we refer to [2]. Here we show some flaws present in the definition. First we recall the definition of Cross-Domain Anonymity from [2].

Definition 2. *A domain-specific pseudonymous signature scheme $\mathcal{NYMS} = (\mathsf{NymKGen}, \mathsf{NymDSGen}, \mathsf{NymSig}, \mathsf{NymVf})$ is (n, d, t, Q, ϵ) cross-domain anonymous with $Q = (q_c, q_s, q_t)$ is for any algorithm $\mathcal{A}$ running in time t, and making at most q_c queries to the corruption oracle, q_s queries to the signing oracle, and q_t queries to the left-or-right oracle, the probability that the following experiment returns 1 is at most ϵ:*

```
$\mathsf{CD\text{-}Anon}_{\mathcal{A}}^{\mathcal{NYMS}}(k, n, d)$:
$b \xleftarrow{\mathcal{R}} \{0, 1\}$
$\mathsf{T}, \mathsf{S}, \mathsf{W}, \mathsf{B}, \mathsf{C}, \mathsf{N}, \mathsf{D} \leftarrow \emptyset$
$(\mathsf{gpk}, \mathsf{gmsk}, \mathsf{gsk}[\mathsf{nym}]_n, [\mathsf{nym}]_n, \mathsf{dpk}_d) \leftarrow \mathsf{NymKGen}(1^k, 1^n, 1^d)$
$\mathsf{N} = \mathsf{nym}_n$, and $\mathsf{D} = \mathsf{dpk}_d$
$\mathsf{W} = \{\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}) | \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
$\mathsf{W} \odot \mathsf{D} := \{(\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}), \mathsf{dpk}) | \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
$st \leftarrow \mathcal{A}^{Corrupt}(\mathsf{gpk}, \mathsf{W}, \mathsf{W}\odot, \mathsf{D}, \mathsf{N})$
If $\mathcal{A}$ queries $Corrupt(\mathsf{nym})$ on input $\mathsf{nym} \in \mathsf{N}$
    - set $\mathsf{C} \leftarrow \mathsf{C} \cup \{\mathsf{nym}\}$
    - return $\mathsf{gsk}[\mathsf{nym}]$
$d \leftarrow \mathcal{A}^{\mathsf{B}', \mathsf{NymSig}', LoR}(st)$
If $\mathcal{A}$ queries $\mathsf{B}'(\mathsf{dsnym})$ on input $\mathsf{dsnym} \in \mathsf{W}$
    - set $\mathsf{B} \leftarrow \mathsf{B} \cup \{\mathsf{dsnym}\}$
If $\mathcal{A}$ queries $\mathsf{NymSig}'(\mathsf{dsnym}, \mathsf{dpk}, m)$ on input $\mathsf{dsnym} \in \mathsf{W} \setminus \mathsf{B}$, $\mathsf{dpk} \in \mathsf{D}$, and message $m$
    - set $\mathsf{S} \leftarrow \mathsf{S} \cup \{(\mathsf{dsnym}, \mathsf{dpk}, m)\}$
    - find $\mathsf{nym} \in \mathsf{N}$ such that $\mathsf{dsnym} = \mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk})$
    - return $\mathsf{NymSig}(\mathsf{dsnym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}, m)$
If $\mathcal{A}$ queries $LoR(\mathsf{dsnym}_0, \mathsf{dsnym}_1, \mathsf{dpk}, m)$ on input $\mathsf{dsnym}_0, \mathsf{dsnym}_1 \in \mathsf{W} \setminus \mathsf{B}$, $\mathsf{dpk} \in \mathsf{D}$, and message $m$,
    - set $\mathsf{T} \leftarrow \mathsf{T} \cup \{(\{\mathsf{dsnym}_0, \mathsf{dsnym}_1\}), \mathsf{dpk}, m\}$
    - find $\mathsf{nym}_0$, $\mathsf{nym}_1 \in \mathsf{N} \setminus \mathsf{C}$ such that $\mathsf{dsnym}_i = \mathsf{NymDSGen}(\mathsf{nym}_i, \mathsf{gsk}[\mathsf{nym}_i], \mathsf{dpk})$ for $i = 0, 1$
    - return $\perp$ if no such $\mathsf{nym}_0$, $\mathsf{nym}_1$ exist, else return $\mathsf{NymSig}(\mathsf{dsnym}_b, \mathsf{gsk}[\mathsf{nym}_b], \mathsf{dpk}, m)$
Return 1 iff

    a) $d = b$ and
```

b) for any $(\{\mathsf{dsnym}_0, \mathsf{dsnym}_1\}, \mathsf{dpk}, m), (\{\mathsf{dsnym}'_0, \mathsf{dsnym}'_1\}, \mathsf{dpk}, m') \in T$ *we have either* $\{\mathsf{dsnym}_0, \mathsf{dsnym}_1\} = \{\mathsf{dsnym}'_0, \mathsf{dsnym}'_1\}$ *or* $\{\mathsf{dsnym}_0, \mathsf{dsnym}_1\} \cap \{\mathsf{dsnym}'_0, \mathsf{dsnym}'_1\} = \emptyset$*, and*

c) for any $(\mathsf{dsnym}, \mathsf{dpk}, m) \in S$ *there is no* dsnym', m' *such that* $(\{\mathsf{dsnym}, \mathsf{dsnym}'\}, \mathsf{dpk}, m') \in T$

The probability is taken over all coin tosses of NymKGen*,* NymSig*, and* $\mathcal{A}$*, and the choice of* b*.*

In short, the goal of the authors is to provide unlinkability of pseudonyms across different domains. As noticed in [3]:

"The left-or-right challenge takes as input two pseudonyms for the same domain and a message and outputs a signature on this message by one of the corresponding users. A simple strategy to win the game, independently of the construction, is to verify this signature using both pseudonyms: it will be valid for only one of them."

and further

"Moreover, in their game, both pseudonyms queried to the challenge oracle are in the same domain, which does not fit the cross-domain anonymity,"

Indeed, the input of the left-or-right oracle consists of two pseudonyms dsnym_0, dsnym_1, a single domain specification dpk and a message m. The oracle will return $\perp$ if any of these pseudonyms does not belong to any user with regards to the domain dpk. Thus, the definition says nothing about linking pseudonyms across different domains.

Unforgeability. The definition of unforgeability from [2] finally refers to the group managers secret key.

"It should be infeasible to create a valid signature on behalf of an honest pseudonym for a previously unsigned message. This should even hold if the adversary knows the group manager's secret key (but not the user's secret key, else trivial attacks would be possible)."

In our opinion, it seems somehow strange to require hardness of forging a signature on behalf of a user by the group manager while it is the group manager who creates the user's secret keys.

Defining such a requirement is actually saying that: "it is infeasible for the group manager to produce signatures on behalf of honest users, although he owns every secret key, since the group manager simply won't forge this signatures".

However, as the authors state that *"adversary may know the group manager's secret key, it must not collaborate with the group manager during generation"* it seems that the aim of the model should be to capture the case, where after generating a user's secret key the group manager simply "forgets" this key. In theory this may be valid if we assume the secret keys are independently generated, and the secret is immediately deleted (e.g. via a hardware mechanism) on the group manager's side. However, if we treat the group manager's secret key as all his secret data including for instance the seeds for a pseudorandom number generator etc., then an adversary may reconstruct the user's secret keys. Hence, it would be necessary to use true randomness each time a user's secret key is created. In practice, in case of an attack on a group manager, it might be that not only the "secret key" leaks but also, all other sensitive information related to the security of the system.

While in many real life situations we take the risk and trust a certain body, the presented definition and discussion attempts to hide this necessary trust assumption. This may awake a lot of justified concerns.

Seclusiveness. This property should guarantee that no adversary should be able to produce a signature for a pseudonym which would be acceptable in the system as a valid domain signature but which cannot be attributed to any legitimate user created in the system.

While the seclusiveness model seems to be adequate, the construction presented in [2] contains a severe weakness, which the authors actually admit:

"By construction, this means that the adversary can thus only corrupt one user, else z becomes known. When considering blacklisting for our construction, we stipulate this below by requiring that the secrets are stored securely in hardware, or, respectively, that the number of corrupt requests q_c is at most 1."

For a powerful adversary it does not make real difference to break into one or two smart cards implementing domain signatures. Thereby, the assumption of 1-seclusiveness is correctly stated in a purely mathematical sense, but quite dubious in the practical sense. When implementing a system like this, we have to assume that certain party may get the system key z and therefore may produce valid identities. So seclusiveness property does not hold for the system from [2] in a very practical sense. It may be regarded as secure against petty criminals with standard technical capabilities for attacks against smart cards.

In our opinion this Achilles' heel of the construction from [2] is a critical issue and disqualifies it for any large scale practical applications, in particular for an implementation on personal eID cards issued by the state authorities.

3 Efficient and Strongly Secure Dynamic Domain-Specific Pseudonymous Signatures for ID Documents

In this section we review the definitions, constructions and security proofs from [3] (which also appear in the PhD dissertation [4]). The authors of this scheme propose a definition for Domain Pseudonymous Signatures which is supposed to capture the dynamic case, i.e. the scenario in which the users may be added dynamically. Moreover, their definition also allows to add new domains to the system. The authors also encounter some flaws in [2] and attempt to patch them. However, we find that their security models and security proofs have also some serious drawbacks and even mistakes. Finally, we give some comments on the construction.

Bellow we recall the definition from [3]. Since the model and, especially, the oracle description are quite complicated we recall them here instead of referring the reader to the original article [3].

Definition 3. *A dynamic domain-specific pseudonymous signature scheme is given by an issuing authority IA, a set of users $\mathcal{U}$, a set of domains $\mathcal{D}$, and the functionalities* {Setup, DomainKeyGen, Join, Issue, NymGen, Sign, Verify, DomainRevoke, Revoke} *as described below. By convention, the users are enumerated here with indices $i \in \mathbb{N}$ and the domains with indices $j \in \mathbb{N}$.*

Setup: *On input a security parameter λ, this algorithm computes global parameters gpk and an issuing secret key isk. A message space $\mathcal{M}$ is specified. The sets $\mathcal{U}$ and $\mathcal{D}$ are initially empty. The global parameters gpk are implicitly given to all algorithms, if not explicitly specified:*

$$(gpk, isk) \leftarrow \mathsf{Setup}(1^\lambda)$$

DomainKeyGen: *On input the global parameters gpk and a domain $j \in \mathcal{D}$, this algorithm outputs a public key dpk_j for the domain j. At the same time an empty revocation list RL_j associated to the domain j is created:*

$$(dpk_j, RL_j) \leftarrow \mathsf{DomainKeyGen}(gpk, j)$$

Join$\leftrightarrow$Issue: *This protocol involves a user $i \in \mathcal{U}$ and the issuing authority IA.* Join *takes as input the global parameters gpk. Issue takes as input the global parameters gpk and the issuing secret key isk. At the end of the protocol, the user i gets a secret key usk_i and the issuing authority IA gets a revocation token rt_i:*

$$usk_i \leftarrow \mathsf{Join}(gpk) \leftrightarrow \mathsf{Issue}(gpk, isk) \rightarrow rt_i$$

NymGen: *On input the global parameters gpk, a public key dpk_j for a domain $j \in \mathcal{D}$ and a secret key usk_i of a user $i \in \mathcal{U}$, this deterministic algorithm outputs a pseudonym nym_{ij} for the user i related to the domain j:*

$$nym_{ij} \leftarrow \mathsf{NymGen}(gpk, dpk_j, usk_i)$$

Sign: *On input the global parameters gpk, a public key dpk_j of a domain $j \in \mathcal{D}$, a secret key usk_i of a user $i \in \mathcal{U}$, a pseudonym nym_{ij} for the user i in the domain j and a message $m \in \mathcal{M}$, this algorithm outputs a signature σ:*

$$\sigma \leftarrow \mathsf{Sign}(gpk, dpk_j, usk_i, nym_{ij}, m)$$

Verify: *On input the global parameters gpk, a public key dpk_j of a domain $j \in \mathcal{D}$, a pseudonym nym_{ij}, a message $m \in \mathcal{M}$, a signature σ and the revocation list RL_j of the domain j, this algorithm outputs a decision $d \in \{\mathtt{accept}; \mathtt{reject}\}$:*

$$d \leftarrow \mathsf{Verify}(gpk, dpk_j, nym_{ij}, m, \sigma, RL_j)$$

Revoke: *On input the global parameters gpk, a revocation token rt_i of a user $i \in \mathcal{U}$ and a list of domain public keys $\{dpk_j\}_{j \in \mathcal{D}' \subseteq \mathcal{D}}$, this algorithm outputs a list of auxiliary informations $\{aux_j\}_{j \in \mathcal{D}' \subseteq \mathcal{D}}$ intended to the subset $\mathcal{D}' \subseteq \mathcal{D}$ of domains:*

$$\{aux_j\}_{i \in \mathcal{D}' \subseteq \mathcal{D}} \leftarrow \mathsf{Revoke}(gpk, rt_i, \{dpk_j\}_{j \in \mathcal{D}' \subseteq \mathcal{D}})$$

DomainRevoke: On input the global parameters gpk, a public key dpk_j of a domain $j \in \mathcal{D}$, an auxiliary information aux_j and the revocation list RL_j of the domain j, this algorithm outputs an updated revocation list RL'_j:

$$RL'_j \leftarrow \mathsf{DomainRevoke}(gpk, dpk_j, aux_j, RL_j)$$

Now we recall the oracle descriptions from [3] on Figure 1. The games involve the following global variables:

- $\mathcal{D}$ is a set of domains,
- $\mathcal{HU}$ is a set of honest users,
- $\mathcal{CU}$ is a set of corrupted users,
- $\mathcal{CH}$ is the set of inputs to the challenge oracle,
- UU is the list of "uncertainty",
- **usk** lists the users' secret keys,
- **rt** lists the revocation tokens,
- **nym** is a list of pseudonyms,
- **dpk** is a list of domain public keys,
- **RL** is the revocation list,
- Σ is a set of signed messages.

Unforgeability

Below we recall the definition of the unforgeability property from [3].
$\mathsf{Unforgeability}_A^{\mathsf{DSPS}}(\lambda)$:

1. $(gpk, isk) \leftarrow \mathsf{DSPS.Setup}(1^\lambda)$
2. $\mathcal{D}, \mathcal{HU}, \mathcal{CU} \leftarrow \{\}$
3. $\mathcal{O} \leftarrow \{\mathsf{AddDomain}(.), \mathsf{WriteRegistractionTable}(.,.), \mathsf{Sign}(.,.,.), \mathsf{SendToUser}(.,.)\}$
4. $(\mathbf{dpk}_*, nym_*, m_*, \sigma_*) \leftarrow A^{\mathcal{O}}(gpk, isk)$
5. Return 1 if all the following statements hold.
   (a) There exists $j \in \mathcal{D}$ such that $dpk_* = \mathbf{dpk}[j]$
   (b) There exists $i \in \mathcal{HU}$ such that $nym_* = \mathbf{nym}[i][j]$, $\mathbf{usk}[i] \neq \perp$ and $\mathbf{rt}[i] \neq \perp$
   (c) $m_* \notin \Sigma[(i,j)]$
   (d) $\mathsf{DSPS.Verify}(gpk, dpk_*, nym_*, m_*, \sigma_*, \{\}) = \texttt{accept}$
   (e) $\mathsf{DSPS.Verify}(gpk, dpk_*, nym_*, m_*, \sigma_*, L) = \texttt{reject}$ where $L :=$ Domain-Revoke(gpk, dpk_*, DSPS.Revoke(gpk, $\mathbf{rt}[i]$, $\{dpk_*\}$, $\{\}$)
   Otherwise, return 0.

The advantage of an adversary A against the Unforgeability game is defined by

$$Adv_{A,\mathsf{DSPS}}^{unforgeability}(\lambda) := \Pr[Unforgeability_A^{\mathsf{DSPS}}(\lambda) = 1]$$

A DSPS scheme achieves unforgeability if, for all polynomial adversaries A, the function $Adv_{A,\mathsf{DSPS}}^{unforgeability}(.)$ is negligible.

```
AddDomain($j$)
- if $j \in \mathcal{D}$, then abort
- $\mathbf{RL}[j] := \{\}$; $All[j] := copy(\mathcal{HU})$
- $\mathbf{dpk}[j] \leftarrow \mathsf{DomainKeyGen}(gpk, j)$
- $\forall i \in \mathcal{HU}$
    • $\Sigma[(i, j)] := \{\}$; $\mathsf{UU}[(i, j)] := \&(All[j])$
    • $\mathbf{nym}[i][j] \leftarrow \mathsf{NymGen}(gpk, \mathbf{dpk}[j], \mathbf{usk}[i])$
- return $\mathbf{dpk}[j]$

CorruptUser($i$)
- if $i \in \mathcal{HU} \cup \mathcal{CU}$, then abort
- $\mathcal{CU} := \mathcal{CU} \cup \{i\}$
- $\mathbf{usk}[i] := \perp$; $\mathbf{nym}[i] := \perp$; $\mathbf{rt}[i] := \perp$
- $dec[IA][i] := \mathsf{cont}$; $state[IA][i] := (gpk, isk)$

Nym($i, j$)
- if $i \notin \mathcal{HU}$ or $j \notin \mathcal{D}$ or $(i, j) \in \mathcal{CH}$, abort
- $\mathsf{UU}[(i, j)] := \{i\}$; $All[j] := All[j] \setminus \{i\}$
- $\forall i' \in \mathcal{HU} \setminus \{i\}$, if $\mathsf{UU}[(i', j)] \neq \&(All[j])$,
  then $\mathsf{UU}[(i', j)] := \mathsf{UU}[(i', j)] \setminus \{i\}$
- return $\mathbf{nym}[i][j]$

NymDomain($j$)
- if $j \notin \mathcal{D}$, then abort
- $result := random\_perm(copy(All[j]))$
- $\forall i \in \mathcal{HU}$,
    • if $\mathsf{UU}[(i, j)] == \&(All[j])$,
        - $\mathsf{UU}[(i, j)] := copy(All[j])$
- $All[j] := \{\}$; return $\{\mathbf{nym}[i][j]\}_{i \in result}$

Sign($i, j, m$)
- if $i \notin \mathcal{HU}$ or $j \notin \mathcal{D}$, then abort
- $\Sigma[(i, j)] := \Sigma[(i, j)] \cup \{m\}$
- return $\mathsf{Sign}(gpk, \mathbf{dpk}[j], \mathbf{usk}[i], \mathbf{nym}[i][j], m)$

ReadRegistrationTable($i$)
- return $\mathbf{rt}[i]$

WriteregistrationTable($i, M$)
- $\mathbf{rt}[i] := M$

AddUser($i$)
  - if $i \in \mathcal{HU} \cup \mathcal{CU}$, then abort
  - $\mathcal{HU} := \mathcal{HU} \cup \{i\}$
  - run $usk \leftarrow \mathsf{Join}(gpk) \leftrightarrow \mathsf{Issue}(gpk, isk) \rightarrow rt$
  - $\mathbf{usk}[i] := usk$; $\mathbf{rt}[i] := rt$
  - $\forall j \in \mathcal{D}$
      • $\Sigma[(i, j)] := \{\}$; $All[i] := All[j] \cup \{i\}$
      • $\mathbf{nym}[i][j] \leftarrow \mathsf{NymGen}(gpk, \mathbf{dpk}[j], \mathbf{usk}[i])$
      • $\mathsf{UU}[(i, j)] := \&(All[j])$

UserSecretKey($i$)
  - if $i \notin \mathcal{HU}$ or $\exists_j \in \mathcal{D}$, s.t. $(i, j) \in \mathcal{CH}$, abort
  - $\mathcal{HU} := \mathcal{HU} \setminus \{i\}$; $\mathcal{CU} := \mathcal{CU} \cup \{i\}$
  - $\forall j \in \mathcal{D}$,
      • $\mathsf{UU}[(i, j)] := \{i\}$; $All[j] := All[j] \setminus \{i\}$
      • $\forall i' \in \mathcal{HU}$, if $\mathsf{UU}[(i', j)] \neq \&(All[j])$,
        then $\mathsf{UU}[(i', j)] := \mathsf{UU}[(i', j)] \setminus \{i\}$
  - return $\{\mathbf{usk}[i], \mathbf{nym}[i]\}$

Revoke($i, \mathcal{D}'$)
  - $\forall i \in \mathcal{D}'$, call $\textsc{DomainRevoke}(i, j)$
  - return $\{\mathbf{RL}[j]\}_{j \in \mathcal{D}'}$

DomainRevoke($i, j$)
  - if $i \notin \mathcal{HU}$ or $j \notin \mathcal{D}$ or $(i, j) \in \mathcal{CH}$, then abort
  - $aux \leftarrow \mathsf{Revoke}(gpk, \mathbf{rt}[i], \{\mathbf{dpk}[j]\})$
  - $\mathbf{RL}[j] \leftarrow \mathsf{DomainRevoke}(\mathbf{dpk}[j], aux, \mathbf{RL}[j])$
  - $\mathsf{UU}[(i, j)] := \{i\}$; $All[j] := All[j] \setminus \{i\}$
  - $\forall i' \in \mathcal{HU} \setminus \{i\}$, if $\mathsf{UU}[(i', j)] \neq \&(All[j])$,
    then $\mathsf{UU}[(i', j)] := \mathsf{UU}[(i', j)] \setminus \{i\}$
  - return $\mathbf{RL}[j]$

NymSig($nym, j, m$)
  - if $i \notin \mathcal{D}$, then abort
  - find $i \in \mathcal{HU}$ such that $\mathbf{nym}[i][j] == nym$
    if no match is found, then abort
  - $\Sigma[(i, j)] := \Sigma[(i, j)] \cup \{m\}$
  - return $\mathsf{Sign}(gpk, \mathbf{dpk}[j], \mathbf{usk}[i], \mathbf{nym}[i][j], m)$
```

```
SendToUser($i, M_{in}$)
  - if $i \in \mathcal{CU}$, then abort; if $i \notin \mathcal{HU}$, then
    $\mathcal{HU} := \mathcal{HU} \cup \{i\}$; $M_{in} := \epsilon$; $\mathbf{usk}[i] := \perp$; $state[i][IA] := gpk$; $dec[i][IA] := cont$
  - $(state[i][IA]), M_{out}, dec[i][IA]) \leftarrow \mathsf{Join}(state[i][IA], M_{in}, dec[i][IA])$
  - if $dec[i][IA] == \mathtt{accept}$, then $\mathbf{usk}[i] := state[IA]$
  - return $(M_{out}, dec[i][IA])$

SendToIssuer($i, M_{in}$)
  - if $i \notin \mathcal{CU}$, then abort
  - $(state[IA][i], M_{out}, dec[IA][i]) \leftarrow \mathsf{DSPS.Issue}(state[IA][i], M_{in}, dec[IA][i])$
  - if $dec[IA][i] == \mathtt{accept}$, then set $\mathbf{rt}[i] := state[IA][i]$
  - return $(M_{out}, dec[IA][i])$

Challenge($b_A, b_B, j_A, j_B, i_0, i_1$)
  - if $i_0 \notin \mathcal{HU}$ or $i \notin \mathcal{HU}$ or $i_0 == i_1$ or $j_A \notin \mathcal{D}$ or $j_B \notin \mathcal{D}$ or $j_A == j_B$, then abort
  - $\forall j \in \{j_A, j_B\}$, $\exists i \in \{i_0, i_1\}$ such that $\{i_0, i_1\} \not\subset \mathsf{UU}[(i, j)]$, then abort
  - $\mathcal{CH} := \{(i_0, j_A), (i_0, j_B), (i_1, j_A), (i_1, j_B)\}$; return $(\mathbf{nym}[i_{b_A}][j_A], \mathbf{nym}[i_{b_B}][j_B])$
```

Fig. 1. Oracle definition from [3]

Comments. The authors of [3] declare as follows:

"Informally, we want that a corrupted authority and corrupted owners of the domains cannot sign on behalf of an honest user."

Unfortunately, the model fails to provide that, since the adversary does not have the possibility to create or corrupt existing domains. To be more specific, the adversary obtains only the $\mathsf{AddDomain}$ oracle which on input takes an identifier j. Moreover, a forgery needs to be done for a domain added by the challenger via the $\mathsf{AddDomain}$ oracle. Thus, the adversary has no control over the creation of the domains. Hence the model does not capture the case where a malicious domain owner forges a signature on behalf of an honest user.

Note that this leaves us in an awkward situation, where the issuer is not trusted, i.e. the adversary controls him, and we want to protect the scheme against forgeries made by a malicious issuer, but we do not protect the scheme against the party (whoever this party might be) which generates the domain descriptions. Thus, the model implicitly assumes that the party which generates the domain descriptions can be trusted.

Let us also comment the security proof, i.e. the proof of Theorem 7 in [3]. Let us recall literally the response phase from this proof:

$\mathsf{Response}$. A play of A eventually gives $(dpk^*, nym^*, m^*, \sigma^*)$. If this is a valid and non trivial response, then (i) we can find a domain j such that $dpk^* = \mathbf{dpk}[j]$ and an honest user i with consistent values $nym_{ij} \leftarrow nym[i][j]$, $(F_i, x_i) \in \mathbf{rt}[i]$ and $(*, A_i, x_i, Z_i) \in \mathbf{usk}[i]$ such that $nym_* = nym_{ij}$, $(F_i, x_i) \in \mathbf{rt}[i]$, and (ii) according to the Σ-unforgeability (Lemma 5, Game 2), we are able to extract a valid certificate (f_*, A_*, x_*, Z_*) where (in particular) $nym_* = h^{f_*} \cdot (dpk_*)^{x_*}$. Since discrete representations in $\mathbb{G}_1$ are unique modulo p, then we have that $f_* = \log_g(F_i)$ (the pseudonym must be valid in a non trivial forgery) and $x_* = x_i$. With probability $1/q_U$ we have $i = \mathsf{i}$, since i is independent of the view of A. This implies that $A_i = A_*$ (a value A is determined by f, x and γ). Therefore $A_* = (g_1 \cdot g_1^{f_*})^{\frac{1}{x_*+\gamma}} = (g_1 \cdot \mathsf{H} \cdot h^{f''_{\mathsf{i}}})^{\frac{1}{x_*+\gamma}}$ and we obtain $\Theta = f_* - f''_{\mathsf{i}}$

The conclusion of the sentence starting with the words *"Since discrete representations in $\mathbb{G}_1$ are unique modulo p"* is simply not true. There are exactly p elements (f_*, x_*) which satisfy the equation $nym_* = h^{f_*} \cdot (dpk_*)^{x_*}$. Therefore, with probability $1 - \frac{1}{p} \approx 1$ we have that $f_* \neq \log_{g_1}(F_i)$ and $x_* \neq x_i$. Thus the reduction is evidently incorrect.

In order to make the proof work, one must assume that the party which generates g_1 and h is trusted. To be more specific, the proof should consider the case where the extractor returns (f_*, x_*) such that $h^{f_*} \cdot (dpk_*)^{x_*} = h^{f_i} \cdot (dpk_*)^{x_i} = nym_*$ and $(f_*, x_*) \neq (f_i, x_i)$, for some i. Now, in case a solver puts the discrete logarithm problem instance into the value h and knows $\log_{g_1}(dpk_*)$, he may compute $\log_{g_1}(h) = r(x_i - x_*)/(f_* - f_i)$.

Nevertheless, the system seems still to be secure in the sense of the unforgeability definition given in [3], but the reduction needs to be done similarly as we have described above, and assuming that the party which generates $g_1, h \overset{\mathcal{R}}{\leftarrow} \mathbb{G}_1$

is trusted, i.e. the adversary does not choose h by himself, what the authors of [3] luckily assume.

Seclusiveness

Below we recall the definition of seclusiveness from [3].
$\mathsf{Seclusiveness}_A^{\mathsf{DSPS}}(\lambda)$:

1. $(gpk, isk) \leftarrow \mathsf{DSPS.Setup}(1^\lambda)$
2. $\mathcal{D}, \mathcal{HU}, \mathcal{CU} \leftarrow \{\}$
3. $\mathcal{O} \leftarrow \{\mathsf{AddDomain}(.), \mathsf{AddUser}(.), \mathsf{CorruptUser}(.), \mathsf{UserSecretKey}(.), \mathsf{Sign}(., ., .), \mathsf{ReadRegistractionTable}(.), \mathsf{SendToIssuer}(., .)\}$
4. $(dpk_*, nym_*, m_*, \sigma_*) \leftarrow A^{\mathcal{O}}(gpk, isk)$
5. Find $j \in \mathcal{D}$ such that $dpk_* := \mathbf{dpk}[j]$. If no match is found, then return 0.
6. Return 1 if for all $i \in \mathcal{U}$, one of the following statements hold.
   (a) $\mathbf{rt}[i] = \perp$
   (b) $\mathsf{DSPS.Verify}(gpk,\ dpk_*,\ nym_*,\ m_*,\ \sigma_*,\ \mathbf{RL}) = \mathtt{accept}$ where $\mathbf{RL} := \mathsf{DSPS.DomainRevoke}(gpk,\ dpk_*,\ aux,\ \mathbf{RL}[j])$ and $aux := \mathsf{DSPS.Revoke}(gpk,\ \mathbf{rt}[i],\ \{dpk_*\})$.

   Otherwise, return 0.

The advantage of an adversary A against the Seclusiveness game is defined by

$$Adv_{A,\mathsf{DSPS}}^{Seclusiveness}(\lambda) := \Pr[Seclusiveness]_A^{\mathsf{DSPS}}(\lambda) = 1]$$

A DSPS scheme achieves seclusiveness if, for all polynomial adversaries A, the function $Adv_{A,\mathsf{DSPS}}^{Seclusiveness}(\lambda)$ is negligible.

Comments. Again in the definition of seclusiveness the adversary is not allowed to choose the domain descriptions for himself. However, unlike for unforgeability, the authors do not make an informal statement that malicious domain owners might be controlled by the adversary. Hence the model does not protect against malicious domains owners.

For seclusiveness we have just one remark which actually also applies to unforgeability. For both seclusiveness and unforgeability the authors make use of the extractor of the signature of knowledge. This extractor is based on the proof of knowledge property of the Σ-protocol and proved in Lemma 3. Let us recall literally the beginning of the proof of Lemma 3 from [3]:

Let us assume that a prover is able to give two valid responses $\vec{s}$, $\vec{s'}$ to two different challenges c, c' given the same values $T, (R_1, R_2, R_3)$. First, by dividing

$$h^{s_f} \cdot \mathsf{dpk}^{s_x} = R_1 \cdot \mathsf{nym}^{c} \text{ by } h^{s'_f} \cdot \mathsf{dpk}^{s'_x} = R_1 \cdot \mathsf{nym}^{c'}$$

we obtain $h^{s_f - s'_f} \cdot \mathsf{dpk} = \mathsf{nym}^{c-c'}$. Since $c \neq c'$, then $c - c' \neq 0 \mod p$, so $c - c'$ is invertible modulo p. Hence $\tilde{f} := (s_f - s'_f)/(c - c')$ and $\tilde{x} := (s_x - s'_x)/(c - c')$ such that $\mathsf{nym} = h^{\tilde{f}} \cdot \mathsf{dpk}^{\tilde{x}}$. Then by dividing

$$\mathsf{nym}^{s_a} \cdot h^{-s_d} \cdot \mathsf{dpk}^{-s_b} = R_2 \text{ by } \mathsf{nym}^{s'_a} \cdot h^{-s'_d} \cdot \mathsf{dpk}^{-s'_b} = R_2$$

we obtain $\mathsf{nym}^{s_a - s'_a} = h^{s_d - s'_d} \cdot \mathsf{dpk}^{s_b - s'_b}$. Substituting $\mathsf{nym} = h^{\tilde{f}} \cdot \mathsf{dpk}^{\tilde{x}}$ gives that $(s_d - s'_d) = \tilde{f} \cdot (s_a - s'_a)$ and $(s_b - s'_b) = \tilde{x} \cdot (s_a - s'_a)$ (which holds if h and dpk are generators of $\mathbb{G}_1$).

The argument *"which holds if h and dpk are generators of $\mathbb{G}_1$"* is clearly invalid. The number of pairs $(\tilde{f}, \tilde{x})$ which may satisfy this equations exactly equals the order of the group. However, the equations $(s_d - s'_d) = \tilde{f} \cdot (s_a - s'_a)$ and $(s_b - s'_b) = \tilde{x} \cdot (s_a - s'_a)$ may indeed hold assuming $\log_h(dpk)$ or $\log_{\mathsf{dpk}}(h)$ are unknown to the prover and the DL problem is hard in $\mathbb{G}_1$. Therefore, the existence of the extractor is conditional or at least no other argument is known by far. This subtlety has some minor impact on the proofs of seclusiveness and unforgeablity. Fortunately, the authors give no control to the adversary over computation of dpk or h in their security model. On the other hand, there is no guarantee, that the trusted party which generates e.g. dpk may not be able to forge signatures on behalf of honest users since the reduction for such case does not work. By now, there has not been shown any attack, which would exploit the fact that the extraction property does not work in such configuration, however, this is unfortunately not covered by the security proof.

Cross-Domain Anonymity

Now, we will take a closer look on the, perhaps most important feature of domain signatures which is Cross-Domain Anonymity.

Below we recall the Cross-Domain Anonymity definition from [3].
$\mathsf{Anonymity}_A^{\mathsf{DSPS}}(\lambda)$:

1. $(gpk, isk) \leftarrow \mathsf{DSPS.Setup}(1^\lambda)$
2. $\mathcal{D}, \mathcal{HU}, \mathcal{CU}, \mathcal{CH} \leftarrow \{\}$
3. $b_A, b_B \xleftarrow{\mathcal{R}} \{0, 1\}$
4. $\mathcal{O} \leftarrow \{\mathsf{AddDomain}(.), \mathsf{AddUser}(.), \mathsf{CorruptUser}(.), \mathsf{UserSecretKey}(.), \mathsf{Revoke}(.,.), \mathsf{DomainRevoke}(.,.), \mathsf{Nym}(.,.), \mathsf{NymDomain}(.), \mathsf{NymSig}(.,.,.), \mathsf{SendToIssuer}(.,.)\}$
5. $b' \leftarrow A^{\mathcal{O}}(gpk)$
6. Return 1 if $b' == (b_A == b_B)$, and return 0 otherwise.

The advantage of an adversary A against the Anonymity game is defined by

$$Adv^{c-d-anon}_{A,\mathsf{DSPS}}(\lambda) := \Pr[Anonymity^{\mathsf{DSPS}}_{A}(\lambda) = 1]$$

A DSPS scheme achieves cross-domain-anonymity if, for all polynomial adversaries A, the function $Adv^{c-d-anon}_{A,\mathsf{DSPS}}(\lambda)$ is negligible.

Comments. In the cross-domain anonymity definition the authors define a uncertainty set for user indexes:

"Since the functionality is dynamic, there might be no anonymity at all if we do not take care of the formalization. For instance, an adversary might ask for adding two domains, two users, i_0, i_1, ask for their pseudonyms through two calls to NymDomain, add a user i_2 and win a challenge involving i_0, i_2 with non-negligible probability. This attack does not work here, since the All list is emptied after each NymDomain call."

Clearly, the authors are right here, however we see the "uncertainty" set mainly as a kind of "smoke grenade" which obscures the view of the model, is not intuitive and makes the model hard to understand.

Moreover, we may consider the following sample game with an adversary A:

1. Start Experiment.
2. A calls $dpk[1] \leftarrow \mathsf{AddDomain}(1)$.
3. A calls $dpk[2] \leftarrow \mathsf{AddDomain}(2)$.
4. A calls $\mathsf{AddUser}(1)$.
5. A calls $\mathsf{AddUser}(2)$.

Note that the adversary did not yet learn any pseudonym or signature of any of the added users in any of the added domains. Now if A calls the challenge oracle $\mathsf{Challenge}(.,.,1,2,1,2)$, i.e. for users 1 and 2, and domains 1 and 2, then the oracle will return $\perp$ since $\{1,2\} \not\subset \mathsf{UU}$. So, the notion of uncertainty set does not allow the adversary for such a strategy, even if the adversary did not ask for any pseudonym or signature. After a careful analysis one can find out that first all users have to be added to the system, and then new domains can be added in order to make the Challenge oracle not to return $\perp$.

Let us now consider consequences of this situation. It seems that according to this definition cross-domain anonymity may only be achieved when there is a set of users added to the system before any domain appears, regardless whether a user creates a signature or publishes a pseudonym or not. So, according to the definition, it seems that only a static group of users determined before creating any domain may stay cross-domain anonymous.

Implementation Issues. The authors main use case is the implementation of their protocol for smart cards, or to be more specific, on identity documents like electronic passports and eID documents. The protocol involves some computations in $\mathbb{G}_T$ which is considered to be heavy for smart cards. Thus, the authors propose to delegate the heavy computation of the smart card to a reader.

The authors claim that

"In our construction, the adversary can compute A from B_2 and σ (if $\sigma = (T, c, s_f, s_x, s_a, s_b, s_d)$, then $A = T \cdot (B_2 \cdot h^{s_a})^{-1/c}$. The fact that we can simulate signatures even in the cross-domain anonymity game shows that the knowledge of A does not help linking users across domains."

Obviously, the fact that a part of the private key leaks is true and worrisome. But the fact that we can simulate the signature, does not imply that the knowledge of A does not help a potential adversary. From their analysis of the "P" protocol on which they base their construction, we see that the honest verifier zero-knowledge property is preserved on common input the global public parameters gpk, a domain specification dpk and a pseudonym nym, where the witness is a tuple (f, A, x) such that $A = (g_1 \cdot h^f)^{1/(\gamma+x)}$ and $nym = h^f \cdot dpk^x$. Clearly, when considering the delegation procedure the protocol is not honest verifier zero knowledge anymore. Some bits of the witness leak! Thus, it remains unclear whether the knowledge of A does not help a potential adversary in linking pseudonyms or forging signatures. It seems that the authors probably also noticed this problem, since in the model extension they restrict the Challenge oracle as follows:

"we restrict the Challenge query to involve at most one user for which the adversary called GetPreComp (before and after the Challenge call)."

Obviously, in the proof of Theorem 9, the user for which GetPreComp was not called, is the one for which the solving algorithm will "inject" the DDH problem instance, and this is exactly the case, where in case such user would delegate the computation, the solving algorithm could not simulate the signature.

Moreover, the model implicitly also assumes that users stay cross-domain anonymous only if they do not delegate the computation.

It seems that in case of unforgeability this argument does not work, since in the proof of Theorem 9 the authors totally change the signature simulation algorithm.

A final practical remark on the delegation approach is that maybe the knowledge of A does not help in linking users across different domains directly. By now, no direct attack has been found and luckily $A \in \mathbb{G}_1$ and $nym \in \mathbb{G}_1$ where DDH is assumed to be hard. However, this may still be considered as a privacy threat. Let us consider the following example. Let us assume that a user creates a signature and the reader is malicious. Thus the reader might collect pairs (A, nym) and (A, nym') for different domains. Of course, there is no proof or check that A really corresponds to nym or nym', however why should a smart card "cheat" and send a different A? This reminds somehow on IP addresses, where obviously a service receiving an IP address of a user does not have any undeniable proof that this particular user used that IP (the address may be spoofed). However, in the scientific literature even this is regarded as a very serious privacy threat and there is a plenty of work on hiding the IP addresses.

4 Conclusions

We have shown some drawbacks in the security model, thus it clearly does not capture what is required from a Domain Pseudonymous Signature Scheme. We believe that the description of the model is complicated and somehow obscure to the extend that it might be even impossible for a non expert to understand the security goals of the proposed scheme. We think that even for experts, the analysis of the definitions, proofs and finally the construction itself, is quite tedious.

Despite our remarks, we think that the constructions itself might be useful. It may turn out that after taking care of some essential detail we could propose a more readable and sound model, and was is more, get rid of the controversial properties the reviewed protocols have.

References

1. Kutyłowski, M., Shao, J.: Signing with multiple id's and a single key. In: Consumer Communications and Networking Conference, IEEE CCNC 2011. (Jan 2011) 519–520
2. Bender, J., Dagdelen, z., Fischlin, M., Kgler, D.: Domain-specific pseudonymous signatures for the german identity card. In Gollmann, D., Freiling, F., eds.: Information Security. Volume 7483 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 104–119
3. Bringer, J., Chabanne, H., Lescuyer, R., Patey, A.: Efficient and strongly secure dynamic domain-specific pseudonymous signatures for id documents. In Christin, N., Safavi-Naini, R., eds.: Financial Cryptography and Data Security. Volume 8437 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2014) 255–272
4. Patey, A.: Techniques cryptographiques pour lauthentification et lidentification biométriques respectant la vie privée (cryptographic techniques for privacy-preserving biometric authentication and identification). TELECOM ParisTech, PhD Thesis (2014)
5. BSI: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token 2.20. Technical Guideline TR-03110-2 (2015)

A Domain Signature Scheme from [2]

The construction of the domain signature scheme from [2] is as follows:

$\mathsf{NymKGen}(1^k, 1^n, 1^d)$: Let $\mathcal{G} = \langle g \rangle$ be a (public) cyclic group of prime order q. We also assume a public hash function H, modeled as a random oracle in the security proofs. Choose $z \in \mathbb{Z}_q$ randomly and calculate $g_1 := g$ and $g_2 := g^z$. Define $\mathsf{gpk} := g_1^x$ for random $x \in \mathbb{Z}_q$. To generate the secrets for the pseudonyms choose n random elements $x_{2,1}, \ldots, x_{2,n}\mathbb{Z}_q$ and calculate $x_{1,i} = x - z \cdot x_{2,i}$ for $i = 1, 2 \ldots, n$. Define $\mathsf{gsk}[i] := (x_{1,i}, x_{2,i})$. By x_j we denote the $x_{j,i}$ when pseudonym i is clear from context. For the domain-parameters pick random $r_1, \cdots, r_d \in_R \mathbb{Z}_q$ and define $\mathsf{dpk}_i := g^{r_i}$ for $i = 1, \ldots, d$. Store z in gmsk. (Note that once the values $\mathsf{gsk}[\cdot]$ have been output resp. given to the users, the group manager deletes them.)

$\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk})$: Compute and output the domain-specific pseudonym $\mathsf{nym}[\mathsf{dpk}] := \mathsf{dpk}^{x_1}$, which is also sometimes denoted as dsnym when nym and dpk are known from context.

$\mathsf{NymSig}(\mathsf{dsnym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}, m)$: Let $a_1 = g_1^{t_1} \cdot g_2^{t_2}$ and $a_2 = \mathsf{dpk}^{t_1}$, for random $t_1, t_2 \in_R \mathbb{Z}_q$. Compute $c = H(\mathsf{dpk}, \mathsf{dsnym}, a_1, a_2, m)$. Let $s1 = t_1 - c \cdot x_1$ and $s_2 = t_2 - c \cdot x_2$. Then, output $\sigma = (c, s_1, s_2)$. (Note that in the Restricted-Identification protocol the user also sends dsnym which we can include here in the signature, in order to match the protocol description.)

$\mathsf{NymVf}(\mathsf{gpk}, \mathsf{dsnym}, \mathsf{dpk}, m, \sigma, \mathsf{B})$: To verify a signature perform the following steps:
1. Parse $(c, s_1, s_2) \leftarrow \sigma$.
2. Let $a_1 = y^c \cdot g_1^{s_1} \cdot g_2^{s_2}$ and $a_2 = \mathsf{dsnym}^c \cdot \mathsf{dpk}^{s_1}$.
3. Output 1 iff $c = H(\mathsf{dpk}, \mathsf{dsnym}, a_1, a_2, m)$ and $\mathsf{dsnym} \notin \mathsf{B}$.

B Domain Signature Scheme from [3]

Below we recall the domain signature scheme from [3].

```
$\mathsf{Setup}(1^\lambda)$:
    1. Generate an asymmetric bilinear environment $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$
    2. Pick generators $g_1, h \xleftarrow{\mathcal{R}} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_T}\}$ and $g_2 \xleftarrow{\mathcal{R}} \mathbb{G}_2 \setminus \{1_{\mathbb{G}_2}\}$
    3. Pick $\gamma \in \mathbb{Z}_p$; Set $w := g_2^\gamma$
    4. Choose a hash function $\mathcal{H} : \{0,1\}^* \leftarrow \{0,1\}^\lambda$
    5. Return $\mathsf{gpk} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, h, g_2, w, \mathcal{H})$; $\mathsf{isk} := \gamma$
$\mathsf{DomainKeyGen}(\mathsf{gpk}, j)$:
    1. Pick $r \xleftarrow{\mathcal{R}} \mathbb{Z}_p^*$; Set $RL_j \leftarrow \{\}$; Return $\mathsf{dpk}_j := g_1^r$; $RL_j$
$\mathsf{Join}(\mathsf{gpk}) \leftrightarrow \mathsf{Issue}(\mathsf{gpk}, \mathsf{isk})$:
    1. $[i]$ Pick $f' \xleftarrow{\mathcal{R}} \mathbb{Z}_p$; Set $F' := h^{f'}$
    2. $[i]$ Compute $\Pi := \mathsf{PoK}\{C := \mathsf{Ext\text{-}Commit}(f') \wedge \mathsf{NIZKPEqDL}(f', C, F', h)\}$
    3. $[U \leftarrow IA]$ Send $F$, $\Pi$    $[IA]$ Check $\Pi$
    4. $[IA]$ Pick $x, f'' \in \mathbb{Z}_p$; Set $F := F' \cdot h^{f''}$; $A := (g_1 \cdot F)^{\frac{1}{\gamma + x}}$; $Z := e(A, g_2)$
    5. $[U \leftarrow IA$ Send $f''$, $A$ , $x$, $Z$
    6. $[i]$ Set $f := f' + f''$; Check $e(A, g_2^x \cdot w) \overset{?}{=} e(g_1 \cdot h^f, g_2)$
    The user gets $\mathsf{usk}_i := (f, A, x, Z)$; The issuer gets $rt_i := (F, x)$
$\mathsf{NymGen}(\mathsf{gpk}, \mathsf{dpk}_j, \mathsf{usk}_i)$:
    1. Parse $\mathsf{usk}_i$ as $(f_i, A_i, x_i, Z_i)$; Return $\mathsf{nym}_{ij} := h^{f_i} \cdot (\mathsf{dpk}_j)^{x_i}$
$\mathsf{Sign}(\mathsf{gpk}, \mathsf{dpk}, \mathsf{usk}, \mathsf{nym}, m)$:
    1. Parse $\mathsf{usk}$ as $(f, A, x, Z)$
    2. Pick $a, r_a, r_f, r_x, r_b, r_d \xleftarrow{\mathcal{R}} \mathbb{Z}_p$; Set $T := A \cdot h^a$
    3. Set $R_1 := h^{h_f} \cdot \mathsf{dpk}^{r_x}$; $R_2 := \mathsf{nym}^{r_a} \cdot h^{-r_d} \cdot \mathsf{dpk}^{-r_b}$
    4. Set $R_3 := Z^{r_x} \cdot e(h, g_2)^{a \cdot r_x - r_f - r_b} \cdot e(h, w)^{r_a}$
    5. Compute $c := \mathcal{H}(\mathsf{dpk} || \mathsf{nym} || T || R_1 || R_2 || R_3 || m)$
    6. Set $s_f := r_f + c \cdot f$; $s_x := r_x + c \cdot x$; $s_a := r_a + c \cdot a$; $s_b := r_b + c \cdot a \cdot x$;
       $s_d := r_d + c \cdot a \cdot f$
    7. Return $\sigma := (T, c, s_f, s_x, s_a, s_b, s_d)$
$\mathsf{Verify}(\mathsf{gpk}, \mathsf{dpk}, \mathsf{nym}, m, \sigma, RL)$:
    1. If $\mathsf{nym} \in RL$, then return $\mathsf{reject}$ and abort.
    2. Parse $\sigma$ as $(T, c, s_f, s_x, s_a, s_b, s_d)$
    3. Set $R_1' := h^{s_f} \cdot \mathsf{dpk}^{s_x} \cdot nym^{-c}$; $R_2' := \mathsf{nym}^{s_a} \cdot h^{-s_d} \cdot \mathsf{dpk}^{-s_b}$
    4. Set $R_3' := e(T, g_2)^{s_x} \cdot e(h, g_2)^{-s_f - s_b} \cdot e(h, w)^{-s_a} \cdot [e(g_1, g_2) \cdot e(T, w)^{-1}]^{-c}$
    5. Compute $c' := \mathcal{H}(\mathsf{dpk}||\mathsf{nym}||T||R_1'||R_2'||R_3'||m)$
    6. Return $\mathsf{accept}$ is $c = c'$, otherwise return $\mathsf{reject}$.
$\mathsf{Revoke}(\mathsf{gpk}, rt_i, \mathcal{D}')$:
    1. Parse $rt_i$ as $(F_i, x_i)$; Return $\{aux_j := F_i \cdot (\mathsf{dpk}_j)^{x_i}\}_{j \in \mathcal{D}'}$
$\mathsf{DomainRevoke}(\mathsf{gpk}, \mathsf{dpk}_j, aux, RL_j)$:
    1. Return $RL_j := RL_j \cup \{aux_j\}$
```