Domain-Specific Pseudonymous Signatures for the German Identity Card

Jens Bender[1] Özgür Dagdelen[2] Marc Fischlin[2] Dennis Kügler[1]

[1] German Federal Office for Information Security (BSI), Germany
[2] Darmstadt University of Technology, Germany

Abstract. The restricted identification protocol for the new German identity card basically provides a method to use pseudonyms such that they can be linked by individual service providers, but not across different service providers (even not malicious ones). The protocol can be augmented to allow also for signatures under the pseudonyms. In this paper, we thus view —and define— this idea more abstractly as a new cryptographic signature primitive with some form of anonymity, and use the term domain-specific pseudonymous signatures. We then analyze the restricted identification solutions in terms of the formal security requirements.

1 Introduction

The protocols designed for the new German identity card [8] secure the communication between the card and the reader resp. the terminal. See Figure 1 for an overview. Initially, the parties run the password-authenticated connection establishment protocol (PACE) to derive a secure channel between the card and the local reader, with the user's consent who needs to enter the password. Then, the card and the remote end point, called the terminal or service provider, run the extended access control protocol (EAC) to authorize mutually and to establish another secure channel between these points. In the card authentication step anonymity is guaranteed by the fact that a large number of cards share the same secret. The security of these protocols has been analyzed in [5, 7, 13].

The overall design of the identity card includes another protocol, called restricted identification. In this optional protocol, card holders can use domain-specific pseudonyms to interact with service providers such that (a) a service provider can recognize pseudonyms of individual cards and use this information for the service (domain-specific linkability), and (b) different service providers cannot link interactions of one user in their respective domains (cross-domain anonymity). Although the concept of restricted identification in [8] —and the Diffie-Hellman based solution— currently only support recognition of pseudonyms, it can be easily extended for more functionality, by allowing users to create signatures under their pseudonyms.

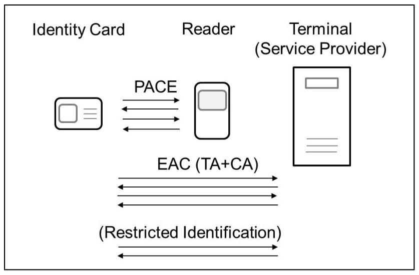

Figure 1: Protocols for the German ID Card

1.1 Domain-Specific Pseudonymous Signatures

To analyze the protocols of the new German identity card we formally introduce the concept of *domain-specific pseudonymous signatures* which captures the scenario of the restricted identification protocols, augmented through signatures, but also allows to reason about the security of the basic restricted identification protocol.[1] In a sense, (domain-specific) pseudonymous signatures can be seen as a relaxed version of group signatures with a limited form of linkability: While group signature schemes, as formalized in [4], provide

[1]Interestingly, the term "pseudonymous signatures" has occasionally been already mentioned in the literature, typically referring to regular signatures under pseudonyms [15], but not considered formally, from a cryptographic point of view.

a very strong form of anonymity, preventing an adversary to identify signers even when knowing the secret keys, pseudonymous signatures are designed specifically to allow a well-defined verifier to link signatures.

The group-signature ancestry also lays the ground to model the security of pseudonymous signatures. As mentioned, according to [4], secure group signature schemes should satisfy full anonymity, the resistance to identify the origin of signatures, and full traceability, the ability of the group manager to trace the origin of a signature with the help of some trapdoor information. The latter property implies, for example, unforgeability (because a forgery could not be traced) and non-frameability, saying that a set of malicious parties, potentially including the group manager, could falsely blame an honest user (which would again contradict traceability).

Concerning anonymity, we note that the goal of restricted identification is exactly to allow a service provider to link previously seen pseudonyms. We thus relax the security requirement for pseudonymous signatures and only demand cross-domain anonymity, i.e., the inability to link signatures given for different service providers, even if both of them are malicious. Furthermore, since the restricted-identification scenario does not involve an authority for traceability, we revert to unforgeability directly for our setting, meaning that one cannot forge signatures on behalf of honest users.

We also introduce another property saying that it is infeasible to make the verifier accept a signature for an invalid domain-specific pseudonym. This assumes a white- or black-listing approach, in the former case making sure that valid pseudonyms are those in the white list, and in the other case demanding that the verifier would not accept a black-listed pseudonym. We observe that this property contrasts with unforgeability which merely protects honest signers from signature forgeries under their name. While this additional property, which we call *seclusiveness*, follows from full-traceability for group signatures, we need to state it explicitly in our setting without a tracing authority. Roughly, unforgeability together with seclusiveness provide a weaker, yet “best-we-can-hope-for” form of full-traceability for domain-specific pseudonymous signatures.

1.2 Analysis of the Restricted-Identification Solutions

We then continue to analyze the current solution for restricted identification (and its signature-augmented version). The basic solution is roughly to give users a random value x_1 and the service provider a certified group element R, such that the user will derive the domain-specific pseudonym as the Diffie-Hellman key $I_R = R^{x_1}$ (or, to be precise, the hash value thereof). This value is then sent over a previously established secure channel to the service provider (but we ignore the channel part in our analysis).

To achieve the additional unforgeability and seclusiveness properties in the signature-augmented case the user is instead given a random representation (x_1, x_2) of the authority's public key y, i.e., $y = g_1^x = g_1^{x_1} g_2^{x_2}$ for generators $g_1, g_2 = g_1^z$. Here, the authority can create such representations easily with the knowledge of $z = \log_{g_1} g_2$. Each domain will again hold a certified group element R, and the user will derive once more the domain-specific pseudonym as the Diffie-Hellman key $I_R = R^{x_1}$ and transmits this value to the provider.

Obviously, if two malicious users could pool their distinct secrets (x_1, x_2) and (x_1', x_2') then they could recover the authority's secret key z. However, we note that these secrets are protected through the hardware of the identity card and are not available to users. In a sense, our security analysis relies on this fact, but at the same time even allows a single (malicious) user access to its secret key. In case of suspicion of leakage of the authority's secret key z a new key can be generated, the old key can be revoked, but white-listing can be used to mark the user keys under the old key as still valid.

In the signature-augmented version, the user additional signs a message m by giving two intertwined non-interactive proofs of knowledge (where the message enters the hash evaluation to derive the challenge non-interactively in the random oracle model). This first proof shows that the user knows the discrete logarithm x_1 of I_R to base R, and the other proof shows that it additionally knows x_2 such that (x_1, x_2) forms a representation of $y = g_1^{x_1} g_2^{x_2}$. The first proof is basically a Schnorr proof of knowledge (PoK) [22], and the second one is an Okamato PoK [18] but re-using the data from the Schnorr proof.

We show that the basic restricted-identification protocol ensures cross-domain anonymity according to our notion. Since pseudonyms $I_R = R^{x_1}$ are essentially Diffie-Hellman keys, this holds under the Decisional Diffie-Hellman assumption. For the signature-augmented version we show that unforgeability holds as well (in the random oracle model, and under the discrete log assumption). This follows from the Schnorr PoK for x_1. Seclusiveness follows from the Okamoto proof showing that the pair (x_1, x_2) is a representation of y and has thus been issued by the authority. Here, however, due to the construction we require that no two malicious users collaborate. As explained above, this is prevented by hiding and protecting the secret of users within the secure hardware of the chip. Thereby, users can use their secrets, however, they cannot extract the keys to collude. Note that the basic version does not implement a signature functionality and thus cannot satisfy these two notions.

1.3 Related Work

As noted above, domain-specific pseudonymous signatures can be seen as descendants of group signatures, only with weaker anonymity requirements but with domain-specific verifiers. The related concept of ring signatures [21] can be viewed as "ad-hoc" group signatures without a central manager. For such schemes, rings of users can be formed at will, and one user can sign on behalf of the ring, but there is usually no mean to identify the actual signer later. As such, domain-specific pseudonymous signatures provide stronger notions of traceability than ring signatures, but a weaker form of anonymity —the domain holder can link signatures.

We note that credential systems [6, 9] are very similar to our pseudonymous signatures, but diverge in some important aspects. Most importantly, credential systems typically provide multi-show unlinkability, as opposed to our pseudonymous signatures. As such, solutions for multi-show credential systems are usually slightly more complex [10–12]. For one-show solutions, where the user can use a credential (under a pseudonym) only once, domain-specific linkability —and therefore cross-domain anonymity— has not been considered before. We also note that the question of turning cross-domain pseudonymous signatures into full-fledged multi-show credential systems is beyond the scope of the paper here: the requirement of recognizing pseudonyms for domain holders is inherent in the application requirement.

Finally, let us point out that our notion of domain-specific pseudonymous signatures is close to a recent proposal of Bernhard et al. [2] for defining direct anonymous attestation (DAA). Their security definition for DAA also resembles the one for group signature schemes but comes with a limited form of linkability: signatures of the same user in the same domain (called base there) should be publicly linkable. At the same time, one should be able to identify signatures given the user's secret key, a requirement which we do not impose in our setting. Moreover, their scenario assumes that linkability must be enforced via cryptographic means, domain-specific pseudonymous signatures allow this by default through the pseudonyms. Besides minor technical differences concerning security in presence of compromised keys or incorporating black-listing, the main difference to their setting is that our model takes into account the extra layer of domain-specific pseudonyms and its unlinkability to the pseudonym layer.

Similarly, Wei [23] also uses DAA as a motivation for his work on tracing-by-linking group signatures, but he considers a weaker form of anonymity where a signer's identity is only hidden if he signs at most a fixed number of times for $k \geq 1$. Any additional signature enables to trace back the identity of the signer by a public algorithm. The approach also includes a notion of k-linkability which corresponds to (public) traceability and essentially means that more than k signatures are linkable and allow to identify the origin. It also contains a notion of non-slanderness which means that no group of malicious users (including the issuing authority) can sign more than k times such that it points to an honest user outside of the group. This primitive does not help in our scenario, because we demand unlinkability only among different domain sectors. Furthermore, our notion of seclusiveness, in contrast with non-slanderness which addresses the linkability of more than k signatures, refers to the fact that one cannot produce any signature on behalf of honest users.

The idea of extending restricted identification to allow unlinkable sector signatures has appeared concurrently in [16]. The scheme does not cover the issue of seclusiveness, though, and is less explicit about the underlying security model, e.g., it remains unclear if unlinkability holds for multiple signatures. Exploring such questions and providing sound models, including issues related to blacklisting or white-listing, and to prove security according to these models for the (augmented) restricted identification protocol of the new German identity cards is our contribution here.

2 Domain-Specific Pseudonymous Signatures

To simplify the notation we denote by $\{a\}_n$ a set of n elements $a_1, \ldots, a_n$ where the indices $i = 1, 2, \ldots, n$ are implicit.

2.1 Preliminaries

Below we define our domain-specific pseudonymous signature scheme first for static groups. Since both the secret keys of users and the domain keys are chosen by the authority, we can imagine that a sufficiently large set is chosen at the outset, and individual entries only become active if required. We hence assume an algorithm $\mathsf{NymKGen}$ generating the group manager's key pair, as well as sufficiently many pseudonyms nym (and secret keys $\mathsf{gsk}[\mathsf{nym}]$) and public domain keys dpk. Each pseudonym and its secret key can now be combined via an algorithm $\mathsf{NymDSGen}$ to build a domain-specific pseudonym $\mathsf{dsnym} = \mathsf{nym}[\mathsf{dpk}]$ which, together with $\mathsf{gsk}[\mathsf{nym}]$ can be used to sign messages.

In addition, we can also introduce the concept of black-lists to the model. For this, we assume that the list is represented by a (dynamically updated) set B of domain-specific pseudonyms which have been revoked earlier. We note that, alternatively, one could employ white-listing where valid pseudonyms are those included in the list; to revoke such a pseudonym one simply removes the entry from the white-list.

Definition 2.1 (Domain-Specific Pseudonymous Signature) *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.*

$\mathsf{NymKGen}(1^\kappa, 1^n, 1^d)$ *is a probabilistic algorithm which, on input a security parameter* 1^κ *and parameters* $1^n, 1^d$ *(both polynomial in* κ*) 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* dpk *to a domain-specific pseudonym* $\mathsf{dsnym} = \mathsf{nym}[\mathsf{dpk}]$*.*

$\mathsf{NymSig}(\mathsf{dsnym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}, m)$ *is a probabilistic algorithm which, on input a domain-specific pseudonym* dsnym*, a secret key* $\mathsf{gsk}[\mathsf{nym}]$*, a domain* dpk*, and message* m*, outputs the signature* σ *of* m *under* $\mathsf{gsk}[\mathsf{nym}]$ *for domain* dpk*.*

$\mathsf{NymVf}(\mathsf{gpk}, \mathsf{dsnym}, \mathsf{dpk}, m, \sigma, \mathsf{B})$ *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 key* dpk*, and a list* B*, outputs either* 1 *(=valid) or* 0 *(=invalid).*

We assume the usual completeness property, that for any honestly generated parameters, domain-specific pseudonyms, and signatures the verification algorithm accepts and outputs 1*.*

Note that we can assume that the group manager uses some standard way of certification for the public keys dpk given out to registered verifiers, and that the signing and verification algorithms check the validity of the keys. We thus often omit this step from the description of protocols.

2.2 Cross-Domain Anonymity

Cross-domain anonymity protects against linking pseudonyms across different domains —within one domain pseudonyms are meant to be linkable. We define cross-domain anonymity via a game between the adversary and a left-or-right oracle which, given two pseudonyms, a message, and a domain, generates a signature for either the left or the right pseudonym in the domain, according to a secret random bit b. We assume that the adversary can make adaptive calls to this left-or-right oracle; this is necessary since we cannot apply a hybrid argument to reduce such multiple queries to a single one (as opposed to the case of full-anonymity for group signatures as in [4], where this is possible since anonymity even holds if the adversary knows the users' secret keys). The adversary's goal is to predict b significantly beyond the pure guessing probability (condition (a) below).

In addition to challenge queries to the left-or-right oracle, the adversary may decide to blacklist domain-specific pseudonyms, to create additional signatures via NymSig, and to corrupt users. For simplicity, we define a version for *static corruptions* where all corruptions are made at the outset, before any other oracle calls are made, and discuss the adaptive version briefly below. To exclude trivial attacks, we must take into account that domain-specific pseudonyms are in principle linkable by the domain holder. Hence, in "transitivity" attacks where the adversary asks the left-or-right oracle first about a signature for domain-specific pseudonyms $\mathsf{dsnym}_0, \mathsf{dsnym}_1$ and then for $\mathsf{dsnym}_0, \mathsf{dsnym}'_1$ for the same domain, but $\mathsf{dsnym}_1 \neq \mathsf{dsnym}'_1$, the signatures would point to the same domain-specific pseudonym if and only if the oracle signs under the left pseudonym. We thus exclude such queries in condition (b) below.

We require another case to be excluded. Namely, the additional signatures generated through NymSig cannot hide the pseudonyms behind the signatures; this would require further means like anonymous signatures and would only work if signatures are not publicly verifiable [14, 24]. Since such extra signatures for domain-specific pseudonyms would thus also allow to link the origin in the left-or-right queries to the pseudonyms, we must disallow the adversary from querying NymSig about domain-specific pseudonyms, which are also used in left-or-right queries (condition (c) below).

The model below assumes, to the advantage of the adversary, that all pseudonyms, all domain keys, and domain-specific pseudonyms are known at the outset; only the assignment of pseudonyms to domain-specific pseudonyms remains hidden. Nonetheless, we assume that the relation of domains and domain-specific pseudonyms is known (see below for the motivation). All this is captured by giving the adversary the corresponding data as sets, with $\mathsf{W} \odot \mathsf{D}$ being the set of all domain-specific pseudonyms dsnym and the corresponding domains dpk. The adversary will thus attack domain-specific pseudonyms from W. As common, we measure the adversary's running time including also all steps of honest parties, and covering both phases of the adversary.

In the definition, we presume that domain-specific pseudonyms are unique within a domain; global uniqueness can then be trivially achieved by attaching the domain key to the pseudonym. Indeed, domain-specific uniqueness will be later ensured by the unforgeability property anyway.

Definition 2.2 (Cross-Domain Anonymity) *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* $Q = (q_c, q_s, q_t)$ *if 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*:*

```
Experiment $\mathsf{CD-Anon}_{\mathcal{A}}^{\mathcal{NYMS}}(\kappa, n, d)$
    $b \xleftarrow{\$} \{0,1\}$
    $\mathsf{T}, \mathsf{S}, \mathsf{W}, \mathsf{B}, \mathsf{C}, \mathsf{N}, \mathsf{D} \leftarrow \emptyset$
    $(\mathsf{gpk}, \mathsf{gmsk}, \{\mathsf{gsk}[\mathsf{nym}]\}_n, \{\mathsf{nym}\}_n, \{\mathsf{dpk}\}_d) \leftarrow \mathsf{NymKGen}(1^\kappa, 1^n, 1^d)$
    $\mathsf{N} = \{\mathsf{nym}\}_n$, and $\mathsf{D} = \{\mathsf{dpk}\}_d$
    $\mathsf{W} = \{\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
    $\mathsf{W} \odot \mathsf{D} := \{(\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}), \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
    $st \leftarrow \mathcal{A}^{\mathsf{Corrupt}}(\mathsf{gpk}, \mathsf{W}, \mathsf{W} \odot \mathsf{D}, \mathsf{N})$
        If $\mathcal{A}$ queries $\mathsf{Corrupt}(\mathsf{nym})$ on input $\mathsf{nym} \in \mathsf{N}$
            – set $\mathsf{C} \leftarrow \mathsf{C} \cup \{\mathsf{nym}\}$
            – return $\mathsf{gsk}[\mathsf{nym}]$
    $d \leftarrow \mathcal{A}^{\mathsf{B}', \mathsf{NymSig}', \mathsf{LoR}}(st)$
        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 $\mathsf{LoR}(\mathsf{dsnym}_0, \mathsf{dsnym}_1, \mathsf{dpk}, m)$ 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 $(\{\mathsf{dsnym}_0, \mathsf{dsnym}_1\}, \mathsf{dpk}, m)$, $(\{\mathsf{dsnym}'_0, \mathsf{dsnym}'_1\}, \mathsf{dpk}, m') \in \mathsf{T}$
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 $(\mathsf{dsnym}, \mathsf{dpk}, m) \in \mathsf{S}$ *there is no* dsnym', m'
such that $(\{\mathsf{dsnym}, \mathsf{dsnym}'\}, \mathsf{dpk}, m') \in \mathsf{T}$.

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

We note that the adversary in our game always accesses oracles through their domain-specific pseudonyms. This is possible since $\mathcal{A}$ knows the set W of such pseudonyms (but not the relation to the pseudonyms nym). Having this list at the outset is motivated by the fact that the adversary can potentially collect such domain-specific pseudonyms when acting as a domain holder, where it gets to learn the domain-specific pseudonyms and signatures under these pseudonyms. Note that this also allows to link domain-specific pseudonyms dsnym to domain keys dpk, hence we give $\mathsf{W} \odot \mathsf{D}$ as additional input. This also implies that we cannot grant the adversary access to another signature oracle which it can provide nym, dpk, m to get a signature for m under the corresponding dsnym; it would be easy to check for the validity of the signature under the domain-specific pseudonym with the help of W and to link dsnym to nym. In other words, one can link pseudonyms to their domain-specific pseudonyms given a signature under the pseudonym for the respective domain.

For an adaptive version, the adversary may interleave Corrupt queries with the other oracle queries arbitrarily. Then we must ensure that no nym in a Corrupt query has appeared in an LoR query before, declaring the adversary to lose if this is the case.

2.3 Unforgeability

Unforgeability of domain-specific pseudonymous signatures follows the basic paradigm for regular signatures: 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). Since for unforgeability we do not need to hide the link between pseudonyms and domain-specific pseudonyms, we assume that the adversary simply knows the tuples (dsnym, nym, dpk) of domain-specific pseudonyms, and corresponding pseudonyms and domain keys. Below we say that the adversary wins if it manages to forge a signature under a domain-specific pseudonym dsnym^* which is potentially derived from *some* pseudonym nym in *some* domain dpk; but the adversary does not to specify these values.

Our notion of unforgeability is weaker than the non-frameability property of group signatures in the sense that, even though the adversary may know the group manager's secret key, it must not collaborate with the group manager during generation. We again consider only the version of static corruptions, although here it is straightforward to capture adaptive corruptions by giving the adversary simply the corruption oracle in the second phase, too.

Definition 2.3 (Unforgeability) *A domain-specific pseudonymous signature scheme* $\mathcal{NYMS} = (\mathsf{NymKGen}, \mathsf{NymDSGen}, \mathsf{NymSig}, \mathsf{NymVf})$ *is* (n, d, t, q, ϵ)*-unforgeable if any algorithm* $\mathcal{A}$*, running in time* t *and making at most* q *signing queries, makes the following experiment output* 1 *with probability at most* ϵ*:*

```
***Experiment*** $\mathsf{Unforge}_{\mathcal{A}}^{\mathcal{NYMS}}(\kappa, n, d)$
    $\mathsf{S}, \mathsf{B}, \mathsf{C}, \mathsf{N}, \mathsf{D} \leftarrow \emptyset$
    $(\mathsf{gpk}, \mathsf{gmsk}, \{\mathsf{gsk}[\mathsf{nym}]\}_n, \{\mathsf{nym}\}_n, \{\mathsf{dpk}\}_d) \leftarrow \mathsf{NymKGen}(1^\kappa, 1^n, 1^d)$
    $\mathsf{N} = \{\mathsf{nym}\}_n$, *and* $\mathsf{D} = \{\mathsf{dpk}\}_d$
    $\mathsf{W} \odot \mathsf{N} \odot \mathsf{D} := \{(\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}), \mathsf{nym}, \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
    $st \leftarrow \mathcal{A}^{\mathsf{Corrupt}}(\mathsf{gpk}, \mathsf{gmsk}, \mathsf{W} \odot \mathsf{N} \odot \mathsf{D})$
        *If* $\mathcal{A}$ *queries* $\mathsf{Corrupt}(\mathsf{nym})$ *on input* $\mathsf{nym} \in \mathsf{N}$
            – *set* $\mathsf{C} \leftarrow \mathsf{C} \cup \{\mathsf{nym}\}$
            – *return* $\mathsf{gsk}[\mathsf{nym}]$
    $(m^*, \sigma^*, \mathsf{dsnym}^*) \leftarrow \mathcal{A}^{\mathsf{B}', \mathsf{NymSig}', \mathsf{Corrupt}}(st)$
        *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 $\mathsf{NymSig}(\mathsf{dsnym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}, m)$
    Output 1 iff there are $\mathsf{nym}^* \in \mathsf{N} \setminus \mathsf{C}$ and $\mathsf{dpk}^* \in \mathsf{D}$ such that
        (a) $\mathsf{NymDSGen}(\mathsf{nym}^*, \mathsf{gsk}[\mathsf{nym}^*], \mathsf{dpk}^*) = \mathsf{dsnym}^*$, and
        (b) $\mathsf{NymVf}(\mathsf{gpk}, \mathsf{dsnym}^*, \mathsf{dpk}^*, m^*, \sigma^*, \mathsf{B}) = 1$, and
        (c) $(\mathsf{dsnym}^*, \mathsf{dpk}^*, m^*) \notin \mathsf{S}$.
```

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

Note that conditions (a) and (c) also imply that domain-specific pseudonyms of different users (within a domain) cannot collide, except with negligible probability. Otherwise, the adversary may corrupt one of the two parties, and any signature created under the domain-specific pseudonym of the one party would immediately constitute a forgery under the other party's pseudonym. (In the above model it would then be more appropriate to let the adversary in calls to NymSig' also specify nym, instead of searching for it; since the adversary knows the list $\mathsf{W} \odot \mathsf{N} \odot \mathsf{D}$ it can look this value up.)

2.4 Seclusiveness

Seclusiveness finally considers the case that the verifier would accept a signature under a domain-specific pseudonym which has not been created by the authority. Note that this assumes that only the black-listed domain-specific pseudonyms are available, but not the universe of all created pseudonyms. This is indeed a valid assumption, following the suggestion in [8] about revocation for pseudonyms through black-lists, with no intention for white-lists. It is also clear that, unlike in case of unforgeability, we thus cannot allow the adversary to know the manager's secret key; else generating keys for additional users would be easy.

As in unforgeability, we again consider only the version of static corruptions. One captures adaptive corruptions by giving the adversary simply the corruption oracle in the second phase, too.

Definition 2.4 (Seclusiveness) *A domain-specific pseudonymous signature scheme* $\mathcal{NYMS} = (\mathsf{NymKGen}, \mathsf{NymDSGen}, \mathsf{NymSig}, \mathsf{NymVf})$ *is* (n, d, t, Q, ϵ)*-secluding with* $Q = (q_c, q_s)$ *if any algorithm* $\mathcal{A}$*, running in time* t *and making at most* q_s *signing queries and* q_c *corruption queries, makes the following experiment output* 1 *with probability at most* ϵ*:*

```
Experiment $\mathsf{Sec}_{\mathcal{A}}^{\mathcal{NYMS}}(\kappa, n, d)$
    $\mathsf{W}, \mathsf{B}, \mathsf{C}, \mathsf{N}, \mathsf{D} \leftarrow \emptyset$
    $(\mathsf{gpk}, \mathsf{gmsk}, \{\mathsf{gsk}[\mathsf{nym}]\}_n, \{\mathsf{nym}\}_n, \{\mathsf{dpk}\}_d) \leftarrow \mathsf{NymKGen}(1^\kappa, 1^n, 1^d)$
    $\mathsf{N} = \{\mathsf{nym}\}_n$ and $\mathsf{D} = \{\mathsf{dpk}\}_d$
    $\mathsf{W} = \{\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
    $\mathsf{W} \odot \mathsf{N} \odot \mathsf{D}$
        $:= \{(\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}), \mathsf{nym}, \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$
    $st \leftarrow \mathcal{A}^{\mathsf{Corrupt}}(\mathsf{gpk}, \mathsf{W} \odot \mathsf{N} \odot \mathsf{D})$
        If $\mathcal{A}$ queries $\mathsf{Corrupt}(\mathsf{nym})$ on input $\mathsf{nym} \in \mathsf{N}$
            – set $\mathsf{C} \leftarrow \mathsf{C} \cup \{\mathsf{nym}\}$
            – return $\mathsf{gsk}[\mathsf{nym}]$
    $(m^*, \sigma^*, \mathsf{dsnym}^*) \leftarrow \mathcal{A}^{\mathsf{B}', \mathsf{NymSig}'}(st)$
        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 $\mathsf{dsnym} \in \mathsf{W} \setminus \mathsf{B}$, $\mathsf{dpk} \in \mathsf{D}$,
        and message $m$
            – find $\mathsf{nym} \in \mathsf{N}$ such that $\mathsf{dsnym} = \mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk})$
            – return $\mathsf{NymSig}(\mathsf{dsnym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}, m)$
    Output 1 iff there exists $\mathsf{dpk}^* \in \mathsf{D}$ such that
        (a) $\mathsf{NymVf}(\mathsf{gpk}, \mathsf{dsnym}^*, \mathsf{dpk}^*, m^*, \sigma^*, \mathsf{B}) = 1$
        (b) $\mathsf{dsnym}^* \notin \mathsf{W}$
```

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

3 Construction

The idea of the discrete-log based construction is as follows: The group manager will hold two generators $g_1, g_2 = g_1^z$ with $z \in \mathsf{gmsk}$ for which it knows the discrete log with respect to each other. In addition, it will hold a public key $y = g_1^x$, such that it can easily compute many pairs (x_1, x_2) such that $y = g_1^{x_1} g_2^{x_2}$ with the help of z; this is the trapdoor property of such values [1,19]. Each user pseudonym nym will receive one of these pairs as its secret key $\mathsf{gsk}[\mathsf{nym}]$. The domain parameters are given by values $\mathsf{dpk} = g^r$. A user can then compute the domain-specific pseudonym as $\mathsf{dsnym} = \mathsf{dpk}^{x_1}$.

To sign a message the user can then use common discrete-log based protocols (in the random oracle model) to show that (a) it knows the discrete-log x_1 of dsnym with respect to dpk, and (b) it knows a matching value x_2 to this discrete logarithm x_1 such that the pair forms a representation of y. Essentially, this is accomplished by running the non-interactive version of the Okamoto proof of knowledge [18] for x_1, x_2 and y to base g_1, g_2, where the x_1-part can simultaneously be used to show knowledge of x_1 of dsnym with respect to base dpk. As usual, the message to be signed enters the hash computations.

Construction 3.1 *The construction of the domain-specific pseudonymous signature scheme* $\mathcal{NYMS} = (\mathsf{NymKGen}, \mathsf{NymDSGen}, \mathsf{NymSig}, \mathsf{NymVf})$ *is as follows:*

$\mathsf{NymKGen}(1^\kappa, 1^n, 1^d)$**:** *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 $z \in_R \mathbb{Z}_q$ randomly and calculate $g_1 := g$ and $g_2 := g^z$. Define $\mathsf{gpk} := g_1^x$ for random $x \in_R \mathbb{Z}_q$. To generate the secrets for the pseudonyms choose n random elements $x_{2,1}, \ldots, x_{2,n} \in_R \mathbb{Z}_q^*$ 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 $r_1, \ldots, r_d \in_R \mathbb{Z}_q^*$ 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 $s_1 = t_1 - c \cdot x_1$ 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 $\mathsf{dsnym} \notin \mathsf{B}$.*

Revocation Mechanisms.

We presented our construction above in terms of black-listing, revoking fraudulent domain-specific pseudonyms by listing them explicitly. Alternatively, and our definitions and constructions are robust in this regard, one can use a white-listing approach to list valid entries only. To represent the whitelisting approach in our framework any delisted domain-specific pseudonym from W will be put in B, such that $\mathsf{W} \setminus \mathsf{B}$ corresponds to the set of currently whitelisted entries. Checking for whitelisting thus corresponds to verifying that the entry is in $\mathsf{W} \setminus \mathsf{B}$ in our framework.

Blacklisting and whitelisting is performed by calculating the domain-specific pseudonym dsnym for domain dpk, but without knowledge of the private keys x_1 and x_2. One important difference between black- and whitelisting is that whitelisting allows to retain security even if authority's secret key z is compromised. If whitelisting is used it is not strictly required to keep z secret. While an attacker would be able to construct valid private keys (x_1, x_2) corresponding to the group public key y, the corresponding pseudonyms would not be listed on the whitelist and thus, the signatures would be rejected. Therefore, the attacker would have to find corresponding private keys for given pseudonyms on the whitelist, which in turn would require to calculate discrete logarithms.

4 Security Analysis

Our proofs work in the random oracle model, and thus, an adversary may also query a random hash function oracle. By q_h, we denote the maximum number of queries to hash function oracle made by the adversary.

4.1 Number-Theoretic Assumptions

Our proof for the anonymity follows by reduction to the Decisional Diffie-Hellman (DDH) assumption whereas the hardness of unforgeability and seclusiveness is reduced to the Discrete Logarithm (DL) problem. Both are considered as standard assumptions, and are believed to be hard for decades. Roughly, the DDH assumption says that distinguishing the tuples (g^a, g^b, g^{ab}) and (g^a, g^b, g') where g' is picked randomly from all group elements, is hard. The DL assumption says it is hard to find a given group $\mathcal{G}$ generated by g and element $A = g^a$. The following definitions capture these assumptions formally.

Definition 4.1 (DDH Hardness) *The Decisional Diffie-Hellman problem (relative to an instance generator) is (t, ϵ)-hard if any algorithm $\mathcal{A}$, running in time t, makes the following experiment output 1 with probability at most $\frac{1}{2} + \epsilon$:*

```
let the instance generator pick a cyclic group $\mathcal{G}$ of order $q$ with a generator $g$
pick $a, b, c \in \mathbb{Z}_q$
let $d \leftarrow \{0,1\}$ and set $Z = g^{ab}$ for $d = 0$ and $Z = g^c$ for $d = 1$
let $d' \leftarrow \mathcal{A}(\mathcal{G}, g, g^a, g^b, Z)$
output 1 iff $d = d'$
```

Definition 4.2 (DL Hardness) *The Discrete Logarithm problem (relative to an instance generator) is (t, ϵ)-hard if any algorithm $\mathcal{A}$, running in time t, makes the following experiment output 1 with probability at most ϵ:*

```
let the instance generator pick a cyclic group $\mathcal{G}$ of order $q$ with a generator $g$
and pick $a \in \mathbb{Z}_q$
let $a' \leftarrow \mathcal{A}(\mathcal{G}, g, g^a)$
output 1 iff $a = a'$
```

4.2 Anonymity

Informally, the protocol is cross-domain anonymous (with respect to static corruptions) because the domain-specific pseudonyms appear to be random to the adversary under the decisional Diffie-Hellman assumption. Below we write $t' \approx t$ to denote the fact that t' is essentially the same running time as t, except for minor administrative overhead.

Theorem 4.3 *Assume the DDH problem is (t, ϵ)-hard. Then the domain-specific pseudonymous signature scheme $\mathcal{NYMS}$ of Section 3 is (n, d, t', Q, ϵ') cross-domain anonymous with $Q = (q_c, q_s, q_t, q_h)$, where $\epsilon' \leq nd\epsilon + \frac{(q_s+q_t+q_h)(q_s+q_t)}{q^2}$ and $t' \approx t$. This holds in the random oracle model.*

Proof. We combine the ideas of pseudorandom synthesizers of Naor and Reingold [17] with the simulation soundness of the non-interactive zero-knowledge proofs (aka. signatures) in the random oracle model. That is, assume the original attack of the adversary on our protocol. In a first game hop we replace the actual signature computations in LoR and NymSig′ queries by simulated signatures (via programming the random oracle). See [20] for details. This strategy is valid if programming the hash values in such computations have not appeared in previous hash queries of the adversary, nor in previous signature queries. However, since the random group elements a_1, a_2 enter the hash evaluations, the probability that an input collision occurs in any of the at most $q_s + q_t$ signature generations, is at most $(q_s + q_t + q_h)(q_s + q_t)/q^2$. Given that no such collision occurs the simulation is perfectly indistinguishable such that the adversary's success probability cannot increase by more than this term.

Note that after the game hop, we can create valid signatures on behalf of users without knowing the secret keys. In the next game hop, we replace the domain-specific pseudonyms dsnym in LoR queries by random group elements (but in a consistent way). That is, whenever we are supposed to use dsnym we instead use a new random element $\mathsf{dsnym}' \leftarrow \mathcal{G}$, unless nym in combination with dpk has been used before, either in a signature request or an LoR query, in which case we use again the previously generated random value dsnym'.

We claim that this hop cannot increase the adversary's success probability noticeable by the DDH assumption. To this end, we briefly recall the notion of a pseudorandom synthesizer in [17]. The pseudorandom synthesizers for the DH pairs is an $a \times b$ matrix, with the rows labeled by values g^{x_i} and the columns labeled by g^{r_j}, and entries at position i, j set to $g^{x_i r_j}$, such that this matrix is indistinguishable from an $a \times b$ matrix of independent and random group elements, even if the row and column labels g^{x_i} and g^{r_j} are given. In a sense, the matrix entries are correlated but still look random. As discussed in [17] this holds in our case under the DDH assumption. In fact, it allows for a reduction to the DDH problem with a loss of a factor ab where, in our case, after the initial corruption, there are at most $ab \leq nd$ entries of honest users.

In the next game hop, we always use the left domain-specific pseudonym dsnym_0 in LoR queries, independently of the value of b. We stress that, in case of $b = 1$, this does not change the adversary's success probability at all. Assume from now on $b = 1$. Note that each LoR query about $(\mathsf{dsnym}_0, \mathsf{dsnym}_1, \mathsf{dpk}, m)$ is answered by a random element, just as it would be for $b = 0$. All other LoR queries involving dpk can only be about the same pair $(\mathsf{dsnym}_0, \mathsf{dsnym}_1)$ in this order, in reverse order $(\mathsf{dsnym}_1, \mathsf{dsnym}_0)$, or for distinct entries. In the first case, we would answer again consistently with the (wrong) random element, in the second case, we would switch to the other random element, and in the third case use an independent random value. This behavior, however, is identical to the case $b = 0$ from the adversary's point of view. Similarly, the adversary cannot make any signature request for $(\mathsf{dsnym}_0, \mathsf{dpk})$ nor $(\mathsf{dsnym}_1, \mathsf{dpk})$ without losing. It follows that such signature requests do not depend on the bit b. Hence, the probability of the experiment returning 1 does not change.

In the final game, the adversary's success probability is independent of b, and the adversary cannot win with probability more than $\frac{1}{2}$. Collecting all probabilities from the game hops yields the claimed bound. □

4.2.1 Anonymity of Restricted Identification.

Recall that in the basic version of the restricted identification protocol, the user merely shows the domain-specific pseudonym dsnym to the service provider (who checks that this value has not been revoked yet). Anonymity of this solution follows from the proof above under the DDH assumption alone, noting that we do not need to simulate the additional signatures in the random oracle model.[2]

4.3 Unforgeability

Theorem 4.4 *Assume the DL problem is (t, ϵ)-hard on $\mathcal{G}$, then the domain-specific pseudonymous signature scheme $\mathcal{NYMS}$ of Section 3 is (n, d, t', Q, ϵ')-unforgeable with $Q = (q_s, q_h)$, where*

$$\epsilon' \approx (2q)^{-1}(q_h - \sqrt{q_h}\sqrt{\delta\epsilon + (2\delta/q)(q_s + q_h)^2 + q_h})$$

and $t' \approx t$ with $\delta = 4ndq^2$. This holds in the random oracle model.

Proof. We are given an adversary $\mathcal{A}$ which wins in the unforgeability game of $\mathcal{NYMS}$, in which $\mathcal{A}$ outputs a fresh signature under domain-specific pseudonym dsnym. Similarly to the proof of Schnorr signatures [20], we leverage the Forking Lemma, in order to obtain two related forgeries $(c, s_1, s_2), (c', s_1', s_2')$ from which we can extract the witness (resp. discrete logarithm) from the challenge. The DL game asks for the discrete logarithm a of an element $A := g_{dl}^a$ from a presumably DL-hard group $\mathcal{G}_{dl}$.

We describe first the framework for interacting with $\mathcal{A}$.

Setup We obtain $(\mathsf{gpk}, \mathsf{gmsk}, \{\mathsf{gsk}[\mathsf{nym}]\}_n, \{\mathsf{nym}\}_n, \{\mathsf{dpk}\}_d)$ by following the $\mathsf{NymKGen}$ algorithm on input $(1^\kappa, 1^n, 1^d)$ except that group $\mathcal{G}$ is chosen as given by the DL game, i.e., $\mathcal{G} := \mathcal{G}_{DH}$. We choose a random nym^* from $\{\mathsf{nym}\}_n$ and set $\mathsf{gsk}[\mathsf{nym}^*] := *$ meaning that $\mathsf{gsk}[\mathsf{nym}^*]$ is unknown. Similarly, we choose random $\mathsf{dpk}^* \in_R \{\mathsf{dpk}\}_d$ and set $\mathsf{dpk}^* = g_{dl}$. We replace all other domain parameters $\mathsf{dpk}_i \in \{\mathsf{dpk}\}_d$ by setting $\mathsf{dpk}_i = g_{dl}^{r_i}$ where r_i is sampled uniformly from $\mathbb{Z}_q^*$. Next, we prepare the set $\mathsf{W} \odot \mathsf{N} \odot \mathsf{D} = \{(\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}), \mathsf{nym}, \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$ for $\mathcal{A}$. However, we replace elements $(\mathsf{NymDSGen}(\mathsf{nym}^*, \mathsf{gsk}[\mathsf{nym}^*], \mathsf{dpk}), \mathsf{nym}^*, \mathsf{dpk}) \in \mathsf{W} \odot \mathsf{N} \odot \mathsf{D}$ by $(A, \mathsf{nym}^*, \mathsf{dpk})$ if $\mathsf{dpk} = \mathsf{dpk}^*$, or else by $(A^{r_i}, \mathsf{nym}^*, \mathsf{dpk})$. We give $\mathcal{A}$ as input $(\mathsf{gpk}, z, \mathsf{W} \odot \mathsf{N} \odot \mathsf{D})$.

[2]The specification actually lets the user send a hash value of dsnym. This does not affect the discussion, though.

Hash Queries We answer with values uniformly sampled from the range of the hash function but keep being consistent and store input/output in a list. In case, we rewind the adversary, we remove all input/output tuples from the list of queries up to the point of rewinding.

Signature Queries Upon input a domain-specific pseudonym dsnym and a message m, if there exists no element $(\mathsf{dsnym}, \mathsf{nym}^*, *) \in \mathsf{W} \odot \mathsf{N} \odot \mathsf{D}$, we perform NymSig on these values honestly and return the output of NymSig. Else, we simulate a signature $\sigma = (c, s_1, s_2)$ on m for domain-specific pseudonym dsnym by the setting the following.

- $s_1, s_2 \xleftarrow{\$} \mathcal{G}$ and $c \xleftarrow{\$} Range(H)$
- $a_1 = y^c \cdot g_1^{s_1} \cdot g_2^{s_2}$ and $a_2 = \mathsf{dsnym}^c \cdot \mathsf{dpk}^{s_1}$
- Program the random oracle such that upon input $(\mathsf{dpk}, \mathsf{dsnym}, a_1, a_2, m)$, output c. If the oracle was queried before on that input, then we abort the simulation.

Corrupt Queries Upon input pseudonym $\mathsf{nym} \neq \mathsf{nym}^*$, we output $\mathsf{gsk}[\mathsf{nym}]$. Upon input pseudonym gpk, we output $z \in \mathsf{gmsk}$

Output At some point, adversary $\mathcal{A}$ outputs a signature $\sigma^* = (c, s_1, s_2)$ on message m under domain-specific pseudonym dsnym^*. If dsnym^* equals A, we rewind $\mathcal{A}$ to the point where he queried the hashing oracle on input $(\mathsf{dpk}, A, y^c \cdot g_1^{s_1} \cdot g_2^{s_2}, A \cdot \mathsf{dpk}^{s_1}, m)$ and output a different but randomly chosen value $c' \neq c$. The Forking Lemma states, that we can produce a second related signature (c', s_1', s_2') where a_1, a_2 are equally defined in both signatures.

Next, we argue that the simulation of $\mathcal{A}$'s environment is perfect, and given a forgery from $\mathcal{A}$, leveraging the Forking Lemma we successfully derive the secret key x_1 from pseudonym nym^* from the two related signatures and consequently, find the discrete logarithm for our challenge.

We follow all steps of $\mathsf{NymKGen}$ compulsory except for the choice of $\mathcal{G}$, nym^* and domain public keys dpk. The group $\mathcal{G}$ is randomly chosen in the unforgeability game of $\mathcal{NYMS}$ as well as $\mathcal{G}_{dl}$ in the DL experiment, and thus, both groups are indistinguishable. The domain descriptions $\{\mathsf{dpk}\}_d$ are chosen from a uniformly distribution in $\mathcal{G}$ (resp. $\mathcal{G}_{dl}$) and since the generator of the DL instance g_{dl} is a random element, setting $\mathsf{dpk}^* = g_{dl}$ and $\mathsf{dpk}_i = g_{dl}^{r_i}$ for random r_i's, is unnoticeable by $\mathcal{A}$. We also pick one random pseudonym nym^* in the hope that $\mathcal{A}$ will forge on this pseudonym. Consequently, we are sure in case it happens that $\mathcal{A}$ does not ask for nym^*'s secret key, otherwise, $\mathcal{A}$ cannot win the unforgeability game. Thus, setting $\mathsf{gsk}[\mathsf{nym}^*] = *$ is again unnoticed by $\mathcal{A}$.

Signature queries can be easily answered due to the fact that we know all secrets but the one of pseudonym nym^*. However, we stress that the simulation described above, outputs a signature on an arbitrary message m under public key dpk for the domain-specific pseudonym dsnym which is shown in Lemma 4.5. We embed our DL challenge in $\mathsf{dsnym}^* := A$ where dsnym^* corresponds to nym^* and dpk^*, again in the hope that $\mathcal{A}$ will forge on behalf of nym^* under domain description dpk^*. This simulation is indistinguishable as A is uniformly distributed as a domain-specific pseudonym dsnym is given that $\mathcal{A}$ does not hold the discrete logarithm of dpk or secret key $\mathsf{gsk}[\mathsf{nym}^*]$, respectively.

Lemma 4.5 *The distribution of* NymSig *and the simulation of signatures as described above is computationally indistinguishable.*

Proof. First, we show the validity of a simulated signature (c, s_1, s_2) and afterwards, we look at the distribution of the underlying elements c, s_1 and s_2.
Validity. The case where dsnym is not derived by nym^* is trivial, since we use the pseudonym's secret key for signing a message. Therefore, we just have to look at the case dsnym corresponding to nym^*.

The verification of a signature $\sigma = (c, s_1, s_2)$ outputs 1, if and only if $c = H(\mathsf{dpk}, \mathsf{dsnym}, y^c \cdot g_1^{s_1} \cdot g_2^{s_2}, \mathsf{dsnym}^c \cdot \mathsf{dpk}^{s_1}, m)$. We have programmed the random oracle such that on input $(\mathsf{dpk}, \mathsf{dsnym}, a_1, a_2, m)$ with $a_1 = y^c \cdot g_1^{s_1} \cdot g_2^{s_2}$ and $a_2 = \mathsf{dsnym}^c \cdot (\mathsf{dpk})^{s_1}$ it outputs c. Indeed, a_1 and a_2 are exactly composed as required. Thus, we have that the oracle outputs the correct hash value c and the verification succeeds.
Distribution. The case where dsnym is not derived by nym^* is again trivial.

The signature consists of elements (c, s_1, s_2). In the signing algorithm NymSig the value c is uniformly drawn from the challenge space, $s_1 = t_1 - c \cdot x_1$ and $s_2 = t_2 - c \cdot x_2$ are uniformly distributed in the group $\mathbb{Z}_q$ because the elements t_1, t_2 are picked randomly from $\mathbb{Z}_q$.

In the simulation, s_1 and s_2 are randomly picked from $\mathbb{Z}_p$, and, therefore, are indistinguishable. We also choose c randomly from the challenge space. This proves what the lemma says. □

We have shown so far, that we provide a perfect simulation of adversary $\mathcal{A}$'s environment if no collision in the signature simulation occurs. This probability is upper bounded by $(2q_s + 2q_h)^2/q$. We still need to show that we leverage the output by $\mathcal{A}$ to solve a hard instance of the DL problem.

In case, $\mathcal{A}$ forges on behalf of pseudonym nym^* under domain dpk^*, we know that the domain-specific pseudonym dsnym is our embedded DL challenge $A = g_{dl}^a$. The probability this event to be happens is at least $1/nd$. A second related forgery can be obtained by using the Forking Lemma where we rewind $\mathcal{A}$ up to the point in which the random oracle was queried on $(\mathsf{dpk}, A, y^c \cdot g_1^{s_1} \cdot g_2^{s_2}, A^c \cdot \mathsf{dpk}^{s_1}, m)$ with the corresponding output c. Now, we give $\mathcal{A}$ a distinct value c' as output by the oracle. Let denote the signature obtained by $\mathcal{A}$ by $\sigma = (c, s_1, s_2)$. Then, Forking Lemma guarantees that we obtain a related forgery $\sigma' = (c', s_1', s_2')$ where the underlying commitments a_1, a_2 and a_1', a_2' (resp. t_1, t_2 and t_1', t_2') are equal with probability $\epsilon'(\epsilon'/q_h + 1/q)$ where ϵ' is the success probability for the forger. Here, we use the bound given by [3]. Note that commitments of our signature schemes are omitted in the final signature for efficiency reasons. In [20], the Forking Lemma was introduced for signatures with commitments being part of the signatures, but the authors mentioned that this is merely for simplicity, and for efficiency reason one might omit the commitment or the challenge, respectively.

Given both signatures σ, σ' we extract the discrete-log a of $\mathsf{dsnym} = A = g^a$ as follows. Given $s_1 = t_1 - c \cdot a$ and $s_1' = t_1 - c' \cdot a$, we have $a = (s_1 - s_1')/(c' - c)$. Hence, we found the solution a for the DL instance A.

We require that $\mathcal{A}$ succeeds to forge on behalf of pseudonym nym^* under domain public key dpk^* in the first signature. In addition, we loose a tightness factor due to Forking Lemma, which yields the probability to find the discrete logarithm of A at most $\epsilon = \epsilon'/nd \cdot (\epsilon'/q_h + 1/q) - (2q_s + 2q_h)^2/q$ where ϵ' is the success probability of $\mathcal{A}$. □

4.4 Seclusiveness

As remarked before, seclusiveness only holds as long as the adversary does not get a hold of the group manager's secret key. 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. If whitelisting is used instead, then we do not require any bound on the number of corruptions the adversary can made, since learning z does not help the adversary to compute a domain-specific dsnym which is listed in the (still trustworthy) whitelist.

Theorem 4.6 *Assume the DL problem is (t, ϵ)-hard on $\mathcal{G}$, then the domain-specific pseudonymous signature scheme of Section 3 is (n, d, t', Q, ϵ')-secluding with $Q = (q_c, q_s, q_h)$, where $q_c = 1$,*

$$\epsilon' \approx (2q)^{-1}(q_h - \sqrt{q_h}\sqrt{q_h + 4q^2\epsilon + 4q^2\delta})$$

and $t' \approx t$ with $\delta = 2(q_s + q_h)/q$. This holds in the random oracle model.

Proof. We are given an adversary $\mathcal{A}$ which wins in the seclusiveness game of $\mathcal{NYMS}$. Here, $\mathcal{A}$ outputs a signature under a domain-specific pseudonym dsnym to no corresponding identity $\mathsf{nym} \in \{\mathsf{nym}\}_d$. Intuitively, the proof works as follows. We are asked for the discrete-log a of an element $A := g_{dl}^a$ from a presumably DL-hard group $\mathcal{G}_{dl}$. We embed A in generator g_2 such that the group's master key $z \in \mathsf{gmsk}$ equals a. We are able to generate one secret key pair (x_1, x_2) satisfying $x = x_1 + zx_2$ for unknown x, gmsk. Using the signature given by $\mathcal{A}$ we can extract a second pair (x_1, x_2). Those two key pairs suffice to disclose z (resp. a) and, thus, we solve the DL problem.

We describe first the framework for interacting with $\mathcal{A}$.

Setup We set $\mathcal{G} = \mathcal{G}_{dl}$, $g_1 = g_{dl}$ and $g_2 = A$. We generate $\{\mathsf{nym}\}_n, \{\mathsf{dpk}\}_d$ following $\mathsf{NymKGen}$. We pick random elements $x_1, x_2 \in_R \mathcal{G}_{dl}$ and set $\mathsf{gpk} = g_1^{x_1} g_2^{x_2}$. After $\mathcal{A}$ selected one pseudonym nym^* to corrupt, we answer by $\mathsf{gsk}[nym^*] := (x_1, x_2)$. We pick n random elements $x_{1,1}, \ldots, x_{1,n}$ and set $\mathsf{gsk}[nym_i] = (x_{1,i}, *)$ for $i = 1, \ldots, n$ with $\mathsf{nym}_i \neq \mathsf{nym}^*$. Next, we prepare the set $\mathsf{W} \odot \mathsf{N} \odot \mathsf{D} = \{(\mathsf{NymDSGen}(\mathsf{nym}, \mathsf{gsk}[\mathsf{nym}], \mathsf{dpk}), \mathsf{nym}, \mathsf{dpk}) \mid \mathsf{nym} \in \mathsf{N}, \mathsf{dpk} \in \mathsf{D}\}$ for $\mathcal{A}$. Note that in our construction we merely need the $x_{1,i}$ to derive the domain-specific pseudonyms. Hence, we give $\mathcal{A}$ as input $(\mathsf{gpk}, \mathsf{W} \odot \mathsf{N} \odot \mathsf{D})$.

Hash Queries The simulation of the random oracle is identical as in the case of unforgeability.

Signature Queries The simulation of signatures is identical as in the case of unforgeability. Again we program the random oracle such that upon input $(\mathsf{dpk}, \mathsf{dsnym}, a_1, a_2, m)$, output c. If the oracle was queried before on that input, then we abort the simulation.

Output At some point, adversary $\mathcal{A}$ outputs a signature $\sigma^* = (c, s_1, s_2)$ on message m under domain-specific pseudonym dsnym^*. If dsnym^* equals A, we rewind $\mathcal{A}$ to the point where he queried the hashing oracle on input $(\mathsf{dpk}, A, y^c \cdot g_1^{s_1} \cdot g_2^{s_2}, A^c \cdot \mathsf{dpk}^{s_1}, m)$ and output a different but randomly chosen value $c' \neq c$. The Forking Lemma states, that we can produce a second related signature (c', s_1', s_2') where a_1, a_2 are equally defined in both signatures.

Next, we argue that the simulation of $\mathcal{A}$'s environment is perfect, and given a forgery from $\mathcal{A}$, leveraging the Forking Lemma, we successfully derive a secret key (x_1^*, x_2^*) such that $x = x_1^* + zx_2^*$ from the two related signatures and consequently, find the discrete logarithm $z = a$ for our challenge A.

The group $\mathcal{G}$ is randomly chosen in the seclusiveness game of $\mathcal{NYMS}$ as well as $\mathcal{G}_{dl}$ in the DL experiment, and thus, both groups are indistinguishable. The values $\{\mathsf{nym}\}_n, \{\mathsf{dpk}\}_d$ are computed compulsory. The group public key gpk is obtained by selecting one secret pair in advance. All other secret keys cannot be specified since z is unknown and thus, we cannot produce key pairs. However, we can select randomly one part $x_{1,i}$, and leave the second part $x_{2,i}$ as unknown.

Our simulation above produces signatures indistinguishable from genuine signatures. We posses the first part of the secret keys, such that we are able to compute the domain-specific pseudonyms. The validity and distribution of the simulated signatures are proven in Lemma 4.5.

We have shown so far, that we provide a perfect simulation of adversary $\mathcal{A}$'s environment if no collision in the signature simulation occurs. This probability is upper bounded by $(2q_s + 2q_h)^2/q$. We still need to show that we leverage the output by $\mathcal{A}$ to solve a hard instance of the DL problem.

At some point, adversary $\mathcal{A}$ outputs a signature $\sigma = (c, s_1, s_2)$ under domain-specific pseudonym dsnym. A second related forgery can be obtained by using the Forking Lemma where we rewind $\mathcal{A}$ up to the point in which the random oracle was queried on $(\mathsf{dpk}, \mathsf{dsnym}, y^c \cdot g_1^{s_1} \cdot g_2^{s_2}, \mathsf{dsnym}^c \cdot \mathsf{dpk}^{s_1}, m)$ with the corresponding output c. Now, we give $\mathcal{A}$ a distinct value c' as output by the oracle. The Forking Lemma guarantees that we obtain a correlated forgery $\sigma' = (c', s_1', s_2')$ where the underlying commitments a_1, a_2 and a_1', a_2' are equal. Given both signatures σ, σ' we extract the discrete-log a of $g_2 = A = g_1^a$ as follows. Given

$$s_1 = t_1 - c \cdot x_1^* \qquad s_1' = t_1 - c' \cdot x_1^* \qquad s_2 = t_2 - c \cdot x_2^* \qquad s_2' = t_2 - c' \cdot x_2^*$$

we have $x_1^* = (s_1 - s_1')/(c' - c)$ and $x_2^* = (s_2 - s_2')/(c' - c)$. Hence, we found a second key pair (x_1^*, x_2^*) satisfying $x = x_1^* + zx_2^*$. We obtain z by $z = (x_1^* - x_1)/(x_2 - x_2^*)$ where x_1, x_2 is the secret key pair we chose in the setup phase for pseudonym nym^*. Hence, since $z = a$, we have found the solution a for the DL instance A. We require that $\mathcal{A}$ succeeds to forge twice, which yields the probability to find the discrete logarithm of A at most $\epsilon = \epsilon'(\epsilon'/q_h + 1/q) - (2q_s + 2q_h)^2/q$ where ϵ' is the success probability of $\mathcal{A}$. $\square$

Acknowledgments

This work was supported by the German Federal Office for Information Security (BSI) and CASED (`http://www.cased.de`). The content and views expressed in this article are those of the authors and do not necessarily reflect the position or policy of the supporting authorities.

References

[1] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[2] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N.P. Smart, and B. Warinschi. Anonymous attestation with user-controlled linkability. Cryptology ePrint Archive, Report 2011/658, 2011. `http://eprint.iacr.org/`.

[3] Mihir Bellare, and Gregory Neven. Multi-signatures in the plain public-Key model and a general forking lemma. In *CCS 2006*, *ACM*.

[4] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629.

[5] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In *ISC 2009*, volume 5735 of *LNCS*, pages 33–48. Springer, September 2009.

[6] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. The MIT Press, 2000.

[7] Christina Brzuska, Özgür Dagdelen, and Marc Fischlin. TLS, PACE, and EAC: A Cryptographic View at Modern Key Exchange Protocols. In *GI-Sicherheit 2012*, *GI-LNI*, pages 71–82. 2012.

[8] Advanced security mechanism for machine readable travel documents extended access control (eac). Technical Report (BSI-TR-03110) Version 2.05 Release Candidate, BSI, 2010.

[9] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28, October 1985.

[10] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118.

[11] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76.

[12] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, *LNCS*, pages 56–72.

[13] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In *ISC 2010*, volume 6531 of *LNCS*, pages 54–68. Springer, October 2010.

[14] Marc Fischlin. Anonymous signatures made easy. In *PKC 2007*, volume 4450 of *LNCS*, pages 31–42.

[15] C.f. B. J. Koops, H. Buitelaar, and M. Lips eds. D5.4: Anonymity in electronic government: a case-study analysis of governments? identity knowledge. *FIDIS report*, Feb 2012.

[16] Miroslaw Kutylowski and Jun Shao. Signing with multiple ID's and a single key. In *38th CCNC*, pages 519–520. IEEE, 2011.

[17] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE, 1997.

[18] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO'92*, *LNCS*.

[19] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140.

[20] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[21] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565.

[22] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[23] Victor K. Wei. Tracing-by-Linking Group Signautres. Cryptology ePrint Archive, Report 2004/370, 2004. `http://eprint.iacr.org/`.

[24] Guomin Yang, Duncan S. Wong, Xiaotie Deng, and Huaxiong Wang. Anonymous signature schemes. In *PKC 2006*, *LNCS*, pages 347–363.