The polymorphic eID scheme

— *combining federative authentication and privacy* —

Eric R. Verheul

Logius
Ministry of Interior and Kingdom Relations
P.O. Box 96810 2509 JE The Hague The Netherlands
eric.verheul@logius.nl

Version 1.2 (preview), 18th September 2019

Abstract We introduce polymorphic encryption and pseudonymisation based on homomorphic properties of ElGamal encryption. We then develop a federative eID scheme with special security, privacy and usability properties. These properties include end-to-end security and privacy between authentication provider and service provider regardless of intermediate parties (proxies). The polymorphic card application envisioned on national eID cards is the scheme showpiece. It allows federative authentication of users in an anonymous way while still providing public service providers the social security number. This novel and paradoxical sounding property is solving a fundamental privacy problem in federated authentication schemes. The scheme also supports "conventional" authentication providers that all produce the same service provider pseudonyms without knowing them. We indicate how this federation can provide self-contained assertions supporting legally binding remote signing sensu the European eIDAS regulation [18], privacy friendly data exchange between service providers, attribute providers and authorization providers.

Keywords: federative eID scheme, homomorphic encryption, pseudonyms, privacy enhanced technology

Contents

1 Introduction 1
1.1 Background 1
1.2 eID pilots 3
1.3 Document outline 6
2 Functional description of the polymorphic eID scheme 7
2.1 Ideas behind polymorphic encryption and pseudonymisation 7
2.2 Outline of the polymorphic eID scheme 9
2.3 Outline of the polymorphic card application (PCA) 14
3 Requirements for the polymorphic eID scheme 16
3.1 eID reliability requirements 16
3.2 eID privacy requirements 17
3.3 eID usability requirements 18
4 Cryptographic primitives, notation and conversions 19
4.1 Background 19
4.2 ElGamal encryption 21
4.3 Digital Signature schemes 23
4.4 Verifiable ElGamal decryption 25
4.5 Symmetric encryption (AES-256) 28
4.6 Key derivation functions 28
4.7 Encoding identities as group elements 30
4.8 Keyed mapping of identities to group elements 32
4.9 Keys, cryptograms and versioning 32
4.9.1 Keys and versioning 32
4.9.2 Cryptograms and versioning 34
4.9.3 Timestamps 35
5 Polymorphic cryptographic building blocks 36
5.1 Introduction 36
5.2 Generation of polymorphic forms at BSN-L during activation . . . 37
5.2.1 Generation of Polymorphic Identity (PI) 38
5.2.2 Generation of Polymorphic Pseudonym (PP) 38
5.2.3 Generation of Polymorphic Identity and Pseudonym (PIP) 39
5.2.4 Generation of Direct Encrypted Identity (DEI) 39
5.2.5 Generation of Direct Encrypted Pseudonym (DEP) 40
5.3 Transformation of polymorphic to encrypted forms at APs 41
5.3.1 Generation of Encrypted Identity (EI) 43
5.3.2 Generation of Encrypted Pseudonym (EP) 45
5.4 Validation and decryption at Service Providers 47
5.4.1 Validation and decryption of an Encrypted Identity 48
5.4.2 Validation and decryption of Direct Encrypted Identity . . . 48
5.4.3 Validation and decryption of an Encrypted Pseudonym . . . 49
5.4.4 Validation and decryption of Direct Encrypted Pseudonym 50
5.4.5 PKCS #11 supported decryption at Service Providers 50
5.5 Correctness proofs 52
6 Formalization of the polymorphic eID scheme 57

6.1 Polymorphic Authenticator Activation . . . 57
6.2 Polymorphic Authentication (Transformation) . . . 59
6.3 Polymorphic Authenticator Deactivation . . . 60
7 The polymorphic card application (PCA) . . . 61
7.1 Introduction . . . 61
7.2 PCA Activation . . . 62
7.3 PCA Authentication . . . 67
7.4 PCA Deactivation . . . 69
7.5 PCA Revocation . . . 69
8 Comparison with requirements . . . 70
8.1 eID reliability requirements . . . 70
8.2 eID privacy requirements . . . 73
8.3 eID usability requirements . . . 74
9 Pseudonym conversion . . . 75
10 Key roll-over . . . 77
10.1 Roll-over for Keygroup #1 . . . 81
10.2 Roll-over for Keygroup #2 . . . 81
10.3 Roll-over for Keygroup #3 . . . 82
10.4 Roll-over for Keygroup #4 . . . 83
10.5 Roll-over for Keygroup #5 . . . 87
10.6 Roll-over for Keygroup #6 . . . 88
10.7 Roll-over to all new keys . . . 89
11 References . . . 90
A Glossary of terms and abbreviations . . . 93
B Representing identity strings as byte arrays . . . 96
C Cryptographic keys in use . . . 98
C.1 Key types and algorithms . . . 98
C.2 Derived keys . . . 101
D PCA PIP transformation algorithms . . . 104
E Verifiable Polymorphic Identity and Pseudonym . . . 106
E.1 Problem description . . . 106
E.2 Generation of Verifiable PIP (VPIP) . . . 107
F Introducing and discussing nPA (informative) . . . 110
G PCA indistinguishability (informative) . . . 115
H Extensions and applications (informative) . . . 115
H.1 EI/EP as self-contained, legally binding assertion holders . . . 116
H.2 Privacy friendly data exchange . . . 117
H.3 Polymorphic eID scheme voting as alternative for postal voting . . . 120

List of Tables

1 Possible cases 52
2 De-activation parameters 78
3 eID (Scheme) Keys 80
4 Possible cases in Keygroup #4 roll-over (rows 1-4) 84
5 Mnemonic for key ID of form $XY_{Z[i]}$ 98
6 eID (Scheme) Keys 99
7 Key derivation 102

List of Figures

1 Routing Services 2
2 Activation 3
3 Transformation 4
4 Visualization of polymorphic encryption 8
5 Visualization of polymorphic pseudonymisation 9
6 Activation at the polymorphic BSN-L 11
7 Transformation at a polymorphic authentication provider 12
8 Working of polymorphic card application (PCA) 15
9 Polymorphic identity algorithm composition 53
10 Polymorphic pseudonym algorithm composition 54
11 Direct pseudonym algorithm composition 56
12 nPA (German eID card) 113
13 “Medical Me” pull based activation and data exchange 119
14 “Medical Me” push based activation and data exchange 120
15 Registration as voter abroad at RVA 123
16 Voting website establishment 124
17 Online voting by voter abroad 125
18 Determination of online election result 126

VERSION CONTROL

Version	Date	Description
0.91	2014-7-7	Initial Draft
0.96	2018-2-26	• further scheme formalization • formalization of polymorphic card application • explicit distinction in polymorphic identity and polymorphic pseudonym • addition of OAEP • addition of EC-Schnorr signatures on encrypted identity and pseudonym • further example applications
1.0	2018-6-18	• processed security review comments from the University of Birmingham • included routing services and end-to-end security and privacy properties of polymorphic authentication in this context • changed EC-Schnorr in line with ISO/IEC 14888-3:2016 • added Direct Encrypted Identity • further clarification and removal of typos
1.1	2019-5-27	• made the Verifiable ElGamal Encryption setup consistent with that of EC-Schnorr • inclusion of Verifiable Polymorphic Identity and Pseudonyms (VPIPs) in Appendix E allowing PCA card issuers to validate PIP correctness prior to card personalisation • explicit attention on conversions to byte arrays added in Section 4.1. • further clarification and removal of typos
1.11	2019-7-8	• minor changes • fixed error in DEI decryption description (Algorithm 22) • added two additional conversion algorithms in Section 8.3 • further clarification and removal of typos
1.2	2019-TBD	• added implementational choices on embedding identities in Section 4.8 • added structures for keys, cryptogram and versioning in Section 4.9 and adapted algorithms and protocols reflecting this • added implementational choices on key derivation in Appendix C.2 • added key roll-over techniques in Section 10 • further clarification and removal of typos

1 Introduction

1.1 Background

Dutch citizens can access electronic public services through the popular DigiD authentication system, cf. www.digid.nl. When authentication is successful, DigiD redirects the user to the public service and provides the national citizen service number (BSN). DigiD is essentially based on userid/password corresponding with eIDAS authentication level Low, cf. [18,19]. As this is not sufficient for Dutch electronic government ambitions it is the objective to have a Dutch authentication scheme supporting eIDAS levels Substantial and High. In essence three types of requirements exist for a (national) electronic authentication scheme:

Reliability Reliability does not only imply that the means of authentication (hereafter: *authenticator*) must be sufficiently secure but also that the binding between the user identity (BSN) and the authenticator must be sufficiently strong. That is, identity theft and mistaken identities should be impossible. The latter imposes conditions on registration and delivery processes deployed. Both reliability aspects are addressed in the eIDAS implementing regulation [19].

Privacy friendliness The system must adhere to the principles of the European General Data Protection Regulation (GDPR) [17] including its data minimisation principle: "personal data collection shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed". Data minimisation also applies to Dutch government and global identifiers such as the BSN should only be used when necessary. In other cases pseudonyms should be used if persistent identifiers are required at all. We note that this also a requirement in the upcoming NIST authentication standard [39].

Usability The system must be easy to use for citizens but also for service providers. It should also facilitate *role* support; depending of his role, a user should be able to log on to a service provider under different identifiers (pseudonyms). This, for instance, should be the case when a user authenticates on behalf of another user (representation). Role support can also be considered a privacy requirement.

To reduce the dependency on DigiD, Dutch government also wants to support authenticators of private parties, i.e. authentication providers (APs). This also increases user friendliness as it gives users more choice. Dutch public services are typically based on the national Social Security Number (BSN). This means that the BSN needs to be communicated by APs to the public service providers as part of the authentication. By Dutch privacy laws BSN processing is very restricted disallowing private APs to store the BSN. This implies that a standard federative setup where the AP registers and provides the BSN to service providers is not possible. This will be elaborated on in Sections 1.2 and 2.

To enable deployment of private authentication providers, Dutch government is setting up an eID authentication federation in line with the SAML standard [41]. This is a "hub and spoke" federation where a so-called *proxy* (cf. [41]) resides between the service provider and the authenticaton provider. The user is first redirected to the proxy accompanied with an authentication request. At the proxy the user can choose an authentication provider of his liking. Next the proxy redirects the user and authentication request to the authentication provider. Here the actual authentication takes place. Following this the user, accompanied with the authentication response, is then redirected to the service provider through the proxy.

To expediently deploy (private) authentication providers the proxy also *translates* between the government authentication protocol used by service providers and the authentication protocols used by the (private) authentication providers. The rationale for this is twofold. First of all, it should be possible to quickly add new (innovative) authentication providers to the eID scheme without burden for the service providers. This is particularly important for service providers with small IT department for which changing authentication protocols can present quite a challenge. Secondly, it is assumed that private authentication providers already, and cost efficiently, deploy a certain authentication protocol. It is not considered cost effective to impose authentication providers to implement the government authentication protocol especially when an authentication contract can be short. To this end, the Dutch eID scheme deploys a translating proxy called *routing service* as indicated in Figure 1.

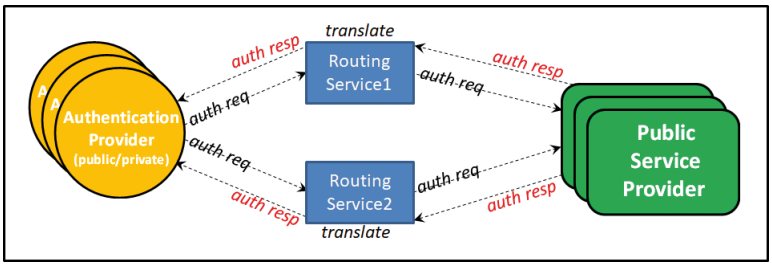

Figure 1. Routing Services

In the routing service setting (or in indeed any proxy setting) two issues regarding privacy and security are essential. First of all, if the routing service has *access* to the actual user data, e.g. the social security number, then the routing service becomes a privacy hotspot that can track all electronic movements of Dutch citizens. Secondly, if the routing service can *change* the actual user data when translating between protocols, it would be in a position to manipulate authentication. That is, an attacker compromising a routing service could systematically substitute one social security number with another allowing fraudulent access to government services for all citizens. Following the eIDAS implementation regulation [19] such an attack on a routing service should not be feasible even by "high potential" attacker. To meet these eIDAS requirements

and to optimally protect against tracking and manipulation attacks at the routing service it is best to have *end-to-end security* and *end-end privacy* between service providers and authentication provider. The first term means that the routing service (or in fact any party between service provider and authentication provider) cannot not successfully manipulate the identity resulting from the authentication. The second term means that the routing service (or in fact any party between service provider and authentication provider) cannot have access to the identity resulting from the authentication. Only the intended service provider has access to this identity. In Section 6.2 we explain how the polymorphic eID scheme can conveniently support both end-to-end security and privacy. We in fact show that even the authentication provider has no access to the identity provided to the service provider, which seems paradoxical. To this end we consider the routing service in essence only as a transport mechanism of encrypted data. For simplicity of presentation we therefore leave out the routing services in this document.

1.2 eID pilots

In the 2016 pilots of the Dutch eID the BSN provisioning issue was addressed by introducing a public attribute provider. This is called the BSN linking service (BSN-L). Coupled with BSN-L are three fundamental use cases: *Activation Transformation* and *Deactivation*. After successful activation the user has an AP authenticator that he can use to authenticate to public service providers providing the BSN. This authentication is a cooperation between the authentication provider and BSN-L whereby the latter performs transformations. Finally, with deactivation a user can arrange that activated authenticators are no longer usable to authenticate to public service providers. Compare Figures 2 and 3.

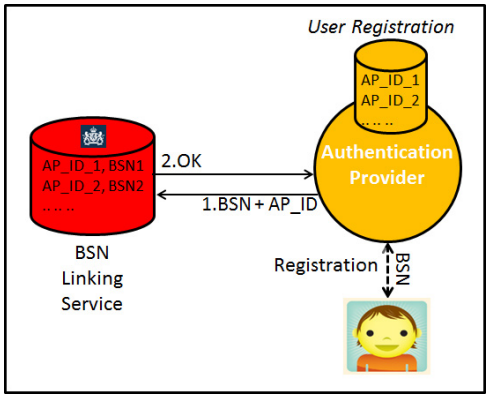

Figure 2. Activation

Authenticator Activation
A user of the (private) AP is provided an authenticator of which the registration and delivery processes comply with the eIDAS regulation [18,19]. The AP temporarily records the user's BSN from its identity document. The AP then activates the user authenticator by sending an AP internal identifier, the BSN and other user identifying data, e.g. first and last names, to BSN-L. Then BSN-L performs some plausibility checks and stores the AP internal identifier and BSN if is this successful. After activation the BSN is deleted by the AP.

Authentication (Transformation)
When the user wants to authenticate to a public service the AP internal identifier needs to be transformed to BSN again. To this end, the user is directed to the AP together with an authentication request. After successful authentication with the AP authenticator, the AP looks up the internal identifier of the user and sends this to BSN-L requesting its BSN for the public service. Then BSN-L looks up up the user's BSN and encrypts this with the public key of the public service. The encrypted BSN is then provided to the AP which sends it to public service in response to the authentication request. The public service then decrypts the BSN and the user is authenticated.

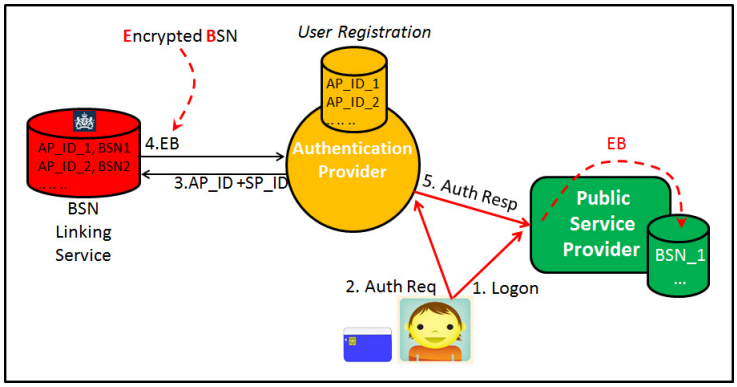

Figure 3. Transformation

Authenticator Deactivation
A user can inform the AP that it no longer wants to use the AP authenticator to authenticate to public service providers. For instance, a user could indicate this by login into in his personal space at the AP and indicate this. The AP then looks up the user internal identifier and sends this to BSN-L requesting the

user entry user to be removed. The user could still use the AP authenticator to authenticate to private service providers. In other words, deactivation is not the same as revocation.

Shortcomings identified in a privacy impact assessment [33] performed on the eID pilots included the following:

1. **BSN-L is a single point of failure** A user cannot authenticate to the government if the BSN-L service is not available. This means that this setup is still dependent on the availability of a central government system. This conflicts with the objectives of Dutch government.
2. **BSN-L is a privacy hotspot** BSN-L can track all movements of Dutch citizens. Indeed, it knows the user identities through their BSNs and the service providers they login to. With the increase of sensitive government services, such as in eHealth care, this is not considered to be a future proof situation. This situation is also in contrast with the data minimisation principle that Dutch government wants to adhere to.
3. **No central user inspection function** There is no centralized mechanism for users to assess where they have registered authenticators. This means that if a fraudster somehow manages to fraudulently register a user at an AP, the user has no simple mechanism to notice that. Only the actual fraud itself can alarm the user.
4. **APs are privacy hotspots** During authentication the AP has both access to the identity of the user as of the service provider the user wants to visit. There are many cases where just registering that a user accessed a specific service constitutes a breach of privacy, e.g. retrieval of medical test results or online psychiatric consultation. This concern grows when private parties with other lines of services, e.g. banks, telcos also take the role of (private) authentication provider. For instance, there might be user perceived influence between the rejection of a mortgage application at the bank and certain regular online medical consults authenticated by the bank.

The third shortcoming can be addressed by a central user inspection service allowing users to see the authentication providers they have registered. Technically this is just a service provider that users can access through the federation. That is, a user can use any activated authenticator to see all of them. Without a central user inspection service, users would be obliged to visit all federative authentication providers which is not workable. The necessity of an inspection service indicates the essence of a central activation step as used in the eID pilots.

The fourth shortcoming appears in any regular federated setup and is explicitly mentioned in the SAML standard [41] on federative authentication. A simple solution would consist of making the authentication provider "blind", i.e. by not giving him the service provider name he is authenticating the user for. Although this appears privacy friendly, this hampers security. Indeed, it would make the system vulnerable to so-called Man-in-the-Browser attacks. That is, malware in the user browser lets the user believe he is authenticating to another service provider then is actually the case. So a user could for instance believe

he logs into a municipality site to make an appointment giving an error message. What actually happens is that malware is surreptitiously logging into the user's online medical file and copies that. More fundamentally, a "blind" authentication provider seems in conflict with privacy regulations. Indeed, a "blind" authentication provider is not able to ask a user for consent to provide data to service providers. We note that article 7 of the GDPR [17] also imposes strict conditions on the form "user consent" takes, including that the controller should be able to demonstrate that consent was given. We also note that the AP is not able to inform the user which service providers he has given data to.

To address these, and other, shortcomings polymorphic encryption and pseudonymisation (PEP) was developed by the author for Dutch government in 2014. A first draft can be found at [48]. The present document contains an updated and enhanced version. The polymorphic setup can also facilitate complete removal of the AP hotspot. Indeed, it allows authentication of a user to a service provider including user consent without the AP knowing the identity of the user. Actually, the AP will not even be able to determine that it performed two authentications for the same person.

1.3 Document outline

This document is meant as a common ground for both non-technical as technical audiences. To accommodate this, the technical detail of the document increases with each section.

- Section 2 contains a functional description of the polymorphic eID scheme and the functional building blocks it is based on. In Subsection 2.3 a functional description of the polymorphic authenticator is given. From a privacy perspective it can be considered the showpiece of the polymorphic setup. It allows federated authentication in practically anonymous way.
- In Section 3 we formalize requirements on the scheme. Sections 2 and 3 are also meant for a non-technical audience.
- Section 4 defines the cryptographic prerequisites required. These are used in Section 5 where detailed cryptographic descriptions of the polymorphic eID scheme building blocks are given that were introduced in Section 2.
- Section 6 formalizes the polymorphic eID scheme introduced in Section 2. The polymorphic authenticator introduced in Subsection 2.3 is formalized in Section 7.
- Section 8 contains a comparison with the requirements from Section 3.
- Section 9 discusses pseudonym conversion.
- Section 10discusses key management. This includes techniques related to various forms of pseudonym conversion and practically convenient algorithms and protocols rolling over to new scheme keys.
- Section 11 contains the references used in this document.

This document also contains a number of appendices:

- Appendix A is a glossary of terms and abbreviatons used throughout the document.

- Appendix B specifies two representations of identity strings as byte arrays.
- Appendix C contains an overview of all cryptographic keys introduced in these specifications.
- Appendix D describes the transformation algorithms performed in context of the Polymorphic Card Application (PCA) which only differ from those in Section 5.3 in that that they do not verify PIP signatures.
- Appendix E describes a technique allowing PCA card issuers validating PIP correctness prior to card personalisation.
- Appendix F is informative only and describes the setup of the German eID-card on which PCA is heavily based.
- Appendix G is also informative only. A PCA user cannot be identified to a group smaller than 20.000. This appendix discusses how this number of 20.000 was arrived.
- Appendix H discusses extensions and applications of the polymorphic setup and has an informative purpose only.

2 Functional description of the polymorphic eID scheme

2.1 Ideas behind polymorphic encryption and pseudonymisation

The eID pilot setup described in Section 1.2 dictates a central user activation step at BSN-L for authentication providers. This allows for a central register where users can inspect the authentication providers they have registered at. When this register is directly fed by BSN-L as part of user activation at the AP this allows for optimal resistance against authentication providers erroneously registering citizens without their knowledge. Our objective is thus to only remove the BSN-L dependency on transformation by enabling authentication providers to perform this themselves. If BSN-L would then not be available, only new users of APs would not be able to activate. Already activated users would still be able to authenticate. This means that the single point of failure at BSN-L is effectively removed. Also, if we allow the authentication providers to perform the transformation themselves, BSN-L no longer is a privacy hotspot either.

To meet this objective, one can let BSN-L return an encrypted BSN as part of activation, i.e. instead of only an “OK” as indicated in Figure 2. However, traditional BSN encryption would imply that only a certain party can decrypt it. Indeed, in traditional data encryption one needs to know the (public) key of the intended recipient prior to encryption. In our context there are many such recipients, namely the service providers. This implies that the BSN would need to be stored at private APs in many encrypted formats too. This leads to a first question: would it be possible to store the BSN in encrypted form at authentication providers such that it can be later transformed into a form decipherable by service provider? During this transformation the BSN should not temporarily emerge at the AP, i.e. a trivial decrypt-encrypt transformation is not fit for purpose. Also, only the intended service provider should be able to decrypt the BSN, i.e. providing all service providers with a same secret key is not fit for purpose either.

As already indicated, the authentication scheme should also support that citizens can authenticate under a pseudonym at government service providers. This should preclude the service providers (in)directly identify the user or link its information to that of other providers. So the user should be provided different pseudonyms at different service providers. Both reliability and user friendliness requirements indicate that such pseudonyms should be independent of the authentication provider the user selected. Indeed, it would not be acceptable if one user could participate several times in an online consultation, i.e. under different pseudonyms. Also, it would not be acceptable if the user's view at a service provider would be dependent of the authentication provider used, i.e. of the pseudonym. A simple setup for this would be to let the pseudonym be cryptographically derived from the BSN and the name of the service provider. However, this again is hampered by the condition that the AP cannot store the BSN. Moreover in this simple setup the authentication provider would know the pseudonym of the user at the service provider. This should be avoided on grounds of the data minimisation principle. This leads to a second question: would it be possible to let authentication providers form compatible pseudonyms for service providers without getting access to them?

Both questions led to the development of polymorphic encryption and pseudonymisation technology. We can picture traditional encryption as putting data in a vault and closing it with a key in possession of the intended recipient. In the polymorphic encryption setup, the encryption is split into two parts: a generic part (using a generic key) performed by BSN-L during activation and a *re-keying* operation at an authentication provider. With re-keying the data is made suitable (decipherable) for the intended recipient by the authentication provider. Here the authentication provider uses different keys corresponding to the intended service providers. See Figure 4 below. The authentication provider will have no access to the plaintext data itself. We similarly have a polymorphic pseudonymisation

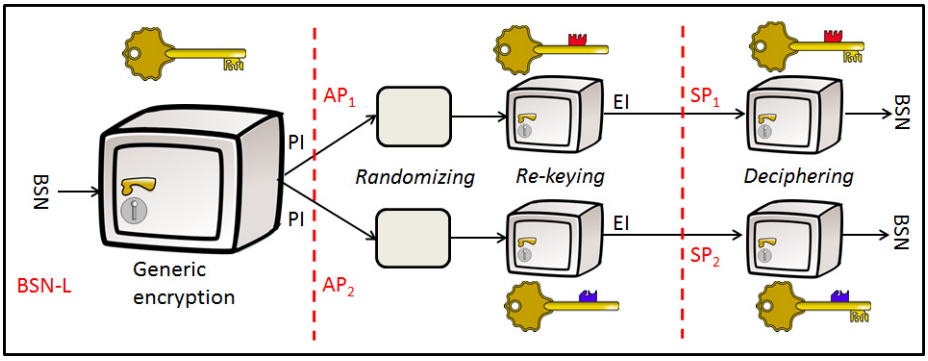

Figure 4. Visualization of polymorphic encryption

setup. Here the rekeying operation is supplemented with a *re-shuffling* operation at the authentication provider. With the latter step a pseudonym is formed "inside" the generically encrypted message without the authentication provider getting access to it. Here the authentication provider also uses different keys corresponding to the intended service providers. See Figure 5. Metaphorically re-shuffling can be compared with shaking the closed vault thereby reordering its contents without knowing the result formed inside. The "shake" corresponds with a cryptographic key and depends of the intended service provider.

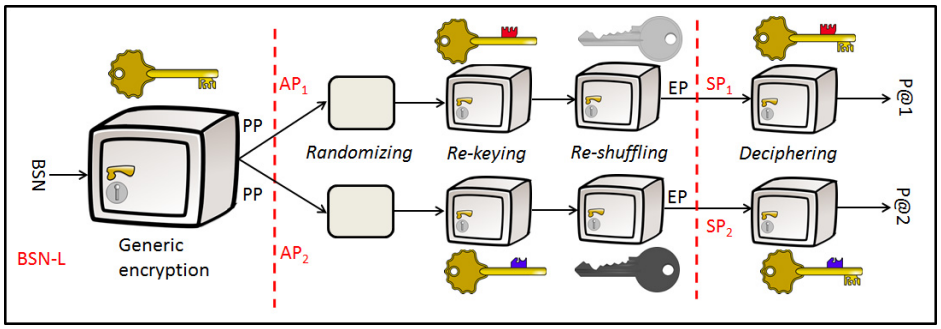

Figure 5. Visualization of polymorphic pseudonymisation

2.2 Outline of the polymorphic eID scheme

From the discussion in the previous session we can now functionally outline the polymorphic eID scheme. As in eID pilot setup of Section 1.2 it is coupled with three fundamental use cases: *Authenticator Activation*, *Authentication (Transformation)* and *Authenticator Deactivation*. Within the polymorphic eID scheme the following participating organisations ("participants") exist:

- BSN-L, associated with a unique identifier BSN-L_{ID}, performing activations.
- Authentication Providers (APs), each associated with a unique identifier AP_{ID} performing transformations as part of authentication.
- Service Providers (SPs), each associated with a unique identifier SP_{ID} providing user services.
- Supervisor, associated with a unique identifier Sup_{ID} that monitors that all parties in the scheme comply with the rules and that participates in dispute handling, e.g. between users, service providers authentication providers. To facilitate the latter each polymorphic cryptogram contains an *Audit Block* holding information on its origin, a time stamp and a sequence number.
- User Inspection Service (UIS), a particular (pseudonymous) service provider where users can view the AP authenticators they have activated. The UIS is

primarily managed by the APs which also arrange that users can distinguish AP authenticators if they have several. However, as part of activation, BSN-L directly informs the user through UIS of an AP activation, i.e. that an AP is activation an authenticator. That is, there should be two UIS user entries for an activated authenticator: one from BSN-L on the AP activating and one from the AP on the authenticator itself. Compare Protocol 1.

Identifiers associated with participants are (non-empty) alphanumeric strings. We also identify a *Key Management Authority* or KMA which provides eID parties with cryptographic keys. The KMA is a trusted third party, i.e. a party that needs to be trusted by all parties and the users of the scheme. For simplicity of terminology, the KMA is not considered a participant of the scheme. We now functionally describe the Activation, Transformation and Deactivation use cases.

Authenticator Activation
Different from the eID pilot situation is that BSN-L only performs cryptographic operations and can be kept stateless. In the polymorphic eID scheme an authentication provider is provided two BSN based cryptographic structures as part of user activation. Compare Step 2 of Figure 6. These structures are *Polymorphic Identities* (PIs) and *Polymorphic Pseudonyms* (PPs). We note that as part of activation, BSN-L can perform various validations before the PI/PP are returned, e.g. if the BSN corresponds to an actual user. BSN-L can also require the authentication provider to provide further information of the user, e.g. first and last name, date of birth birth. This will allow further validations by BSN-L further minimizing activation errors.

The PI is a generically encrypted identity produced by BSN-L. This identity is typically the BSN but the terminology reflects that more "identities" can be used. Specifically one can think of the so-called *uniqueness identifier* from the interoperability framework regulation stipulated in the eIDAS regulation [18]. The PP is a generically encrypted *base pseudonym* produced by BSN-L. The base pseudonym is a keyed hash value of the BSN, cf. Section 5. Whereas the PI can be considered an encrypted BSN, the PP is constructed not to be. This complies with the 2007 pseudonymisation guidelines [15] of the Dutch Data Protection Authority. We also introduce a combination of a PI and PP called *Polymorphic Identity and Pseudonym* (PIP) that saves ⅙-th of data which is convenient in the context of smart cards.

Authentication (Transformation)
Unlike the eID pilot situation transformations are not performed by BSN-L but by the authentication providers themselves. For this authentication providers are provided cryptographic keys by the KMA. These allow AP to transform PIs to *Encrypted Identities* (EIs), i.e. encrypted BSNs. Likewise PPs can be transformed to *Encrypted Pseudonyms* (EPs) for government service providers. This is indicated in Step 5 of Figure 7. Service providers are also provided with decryption keys by KMA allowing to decrypt EIs and EPs. This means that the polymorphic BSN-L no longer is a single point of failure during authentication,

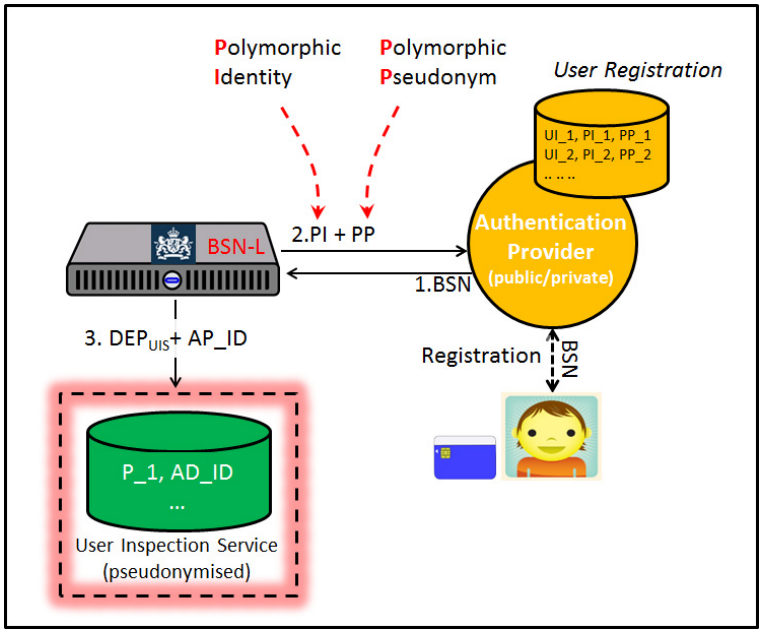

Figure 6. Activation at the polymorphic BSN-L

mitigating the first eID pilot shortcoming from Section 1.2. Moreover, BSN-L is no longer a hotspot and the second eID pilot shortcoming from Section 1.2 is also mitigated.

We have already indicated that authentication providers have no access to the BSN/pseudonyms inside PI/PP. Regardless of this, and in fact regardless of the actual cryptographic implementation, the transformation operation at authentication providers is a sensitive operation. To indicate, if the private keys of one government service provider compromise then with the transformation ability the BSN inside any of his PIs can be retrieved by APs. Indeed, the AP could transform this PI to an EI in the compromised government service provider domain and then decrypt the BSN from it. Of course for this event two failures need to occur: at an authentication provider and at a service provider. This risk is addressed cryptographically by ensuring PI and PP structures are authenticaton provider specific: an authenticaton provider cannot use the PI or PP of another authenticaton provider. To further minimize this risk, KMA, BSN-L and authentication providers keys are controlled in a dedicated device, a so-called Hardware Security Module or HSM. Service providers are not required to use HSMs. The HSMs ensure that cryptographic keys will not be available in plaintext outside the HSM and can only be used in a controlled fashion. With respect to the latter, the generation of a polymorphic form uses several cryptographic keys and the HSM ensures that these keys cannot be used individually

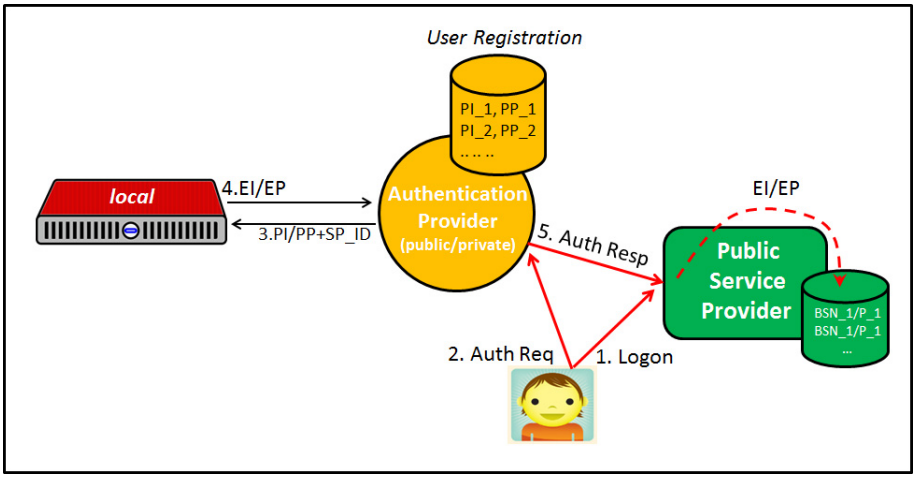

Figure 7. Transformation at a polymorphic authentication provider

and can only be used as part of such a generation. The secure provisioning of HSMs is one of the tasks of the KMA which also holds an HSM registration for this. To this end, each HSM within the scheme has a unique identifier denoted by $\mathrm{HSM}_{\mathrm{ID}}$. The HSMs used cannot be cloned without consent of the KMA. One can compare the HSMs used as an EMV payment card, banks issue to their customers: the customer can use the keys inside the card to form a signed payment request at a payment terminal, but cannot use these keys individually or clone the card.

Note that when the same user wants to authenticate twice to the same service provider, the input (PI/PP and the service provider identity) are the same. If the resulting EIs/EPs would also be the same then this introduces linkability vulnerabilities along the EI/EP transport to the service provider. This vulnerability is addressed by a third operation (next to re-keying and re-shuffling) of the polymorphic scheme: *randomization*. This property allows making a copy of a PI, PP, EI or EP that is not linkable to the original (but contains the same plaintext). Randomization take place in the authentication provider HSM. The three operations encountered so-far (re-key, re-shuffle, randomization) are fundamental for the polymorphic setup and will be cryptographically specified in Section 5.

So far we have only mitigated the first and second eID pilot shortcoming from Section 1.2. The third shortcoming, lack of a central user inspection function, can now also be mitigated. To this end, we introduce a government service that will give users insight in the authentication providers they have activated. Users can access the inspection service like any other government service provider in the federation, i.e. through any authentication provider they have activated. As the BSN is not required in the inspection service it is based on pseudonyms.

The information provisioning of the inspection service should be as independent on the authentication providers as possible. To this end, as part of activation, BSN-L (and not the authentication provider) will produce a message for the inspection service. This message will contain the authentication provider name and an encrypted pseudonym of the user in the inspection domain. This allows the inspection service to register the activation under the user pseudonym. The user can then access this information through any authentication provider in the scheme.

Although, the encrypted pseudonym provided by BSN-L to the user inspection service decrypts to a regular pseudonym of the user it is of special form called *Direct Encrypted Pseudonym* or DEP. The rationale behind this is to ensure that BSN-L, like authentication providers, is not allowed to have access to user pseudonyms. Supporting this requires some further attention. We note that DEPs also allow other functionalities such bootstrapping of attribute providers. For completeness reasons, and not used in the protocols in this document, we also introduce a special form of encrypted identity called *Direct Encrypted Identity* or DEI in Section 5.2.4.

We are left with discussing the mitigation of the last eID pilot shortcoming, i.e. the hotspots at APs. For this the designed eID scheme offers two mitigation solutions. We remark it is important that the AP keeps a records of the service providers visited by the user. These records are vital in case of disputes and should also be accessible by the user. The first mitigation of the shortcoming is requiring authentication providers to separate *user* data, e.g. name address et cetera, from *usage* data, i.e. the identities of the service providers visited by the user. With its pseudonymisation support, the polymorphic eID scheme conveniently caters for this. Indeed, the AP registers an EP for it users at a separate Transaction Log Provider (TLP). This service provider could be a separated part of the AP or could be placed at a different party. As part of authentication, the AP sends the authentication transactions to this service provider coupled with an EP. The user can login to this service providers and view its authentication transactions. This solution does not fully technically enforce separation between user and usage data as a misbehaving AP could still store the transaction together with the user data.

Authenticator Deactivation
As BSN-L is stateless it is not involved in Deactivation. When a user informs an AP to deactivate its authenticator, the AP will no longer allow this authenticator to be used for authentication to public service providers. To make this verifiable for the user, the AP is required to have this reflected in the User Inspection Service. To this end, the AP generates an EP for this service from the user PP and informs the service of deactivation. The AP is also obliged to delete registered polymorphic forms received during activation. As the user can login to user inspection service with any other eID authenticator, he can verify the deactivation. This is also allows him to effectively dispute an authentication performed by the AP on his behalf.

2.3 Outline of the polymorphic card application (PCA)

The polymorphic setup allows for a technical solution of the AP hotspot issue of Section 1.2. This solution technically *enforces* separation between user and usage data at the AP. Here we let the AP place the user polymorphic forms (PI and PP) received during activation in an smart card application called *Polymorphic Card Application* (PCA). The smart card is securely delivered to the user, e.g. in line with the eIDAS High requirements, c.f. [19]. During authentication the user allows the AP reading these polymorphic forms from the card application. However, the card application randomizes these polymorphic forms before delivering them to the AP. In this way, it is ensured that the AP is not able to identify user. Actually, if the user would authenticate again using his PCA card, the AP would not even be able to cryptographically assess that the same user authenticated twice.[1] Moreover, user consent is required (and enforced) in allowing the AP reading PI, PP or both. As explained earlier, the PI enables the AP to authenticate the user whereby providing the BSN to the service provider. Also, the PP part enables the AP to authenticate the user providing a pseudonym to the service provider. In other words, the PCA card gives the user control on what data is provided to the service provider. As will be indicated in Section 7, PCA is a simple extension of the German *neue Personalausweis (nPA)* card application, specifically of its "Restricted Identification" (RI) protocol specified in [5]. This protocol allows an AP the forming and reading of an "RI" pseudonym over a two side secured channel between AP and the eIDAS token. In PCA we do not use the nPA protocols specified for revocation which consists of issuing service provider specific lists of RI pseudonyms corresponding to revoked cards. For revocation (and white listing) we simply used the pseudonyms supported by the polymorphic infrastructure. To this end we introduce a card *status service* (SS) that maintains the status of the card under a pseudonym.

During activation, the card issuer is provided a DEP for the user allowing the issuer to register cards issued and to change card status in the card status service, e.g. as the result of a revocation. Moreover, during authentication, the AP always reads a (randomized) PP from the card and transforms this to EP for a *card status service*. Next, the AP queries the SS and only if the card is issued and not revoked the AP will successfully authenticate the user at the service provider. That is, during authentication the AP reads either both PI and PP from the card (allowing authentication under both BSN and pseudonym) or only the PP from the card (allowing authentication under both BSN and pseudonym).

We finally remark that we use a data efficient combination of PI and PP called Polymorphic Identity and Pseudonym (PIP). A PIP consists of a PI and a PP part which can individually be selected. However a PIP requires only $\frac{5}{6}$-th of data compared to the combined data required for a PI and a PP. In summary: for PCA authentications leading to the BSN the authentication reads

[1] We remark that other information, e.g. IP addresses, browser fingerprints, can allow linking of authentications but this is outside the scope of this document.

a randomized PIP is read and for PCA authentications leading to pseudonyms only a randomized PP is read. Compare Figure 8.

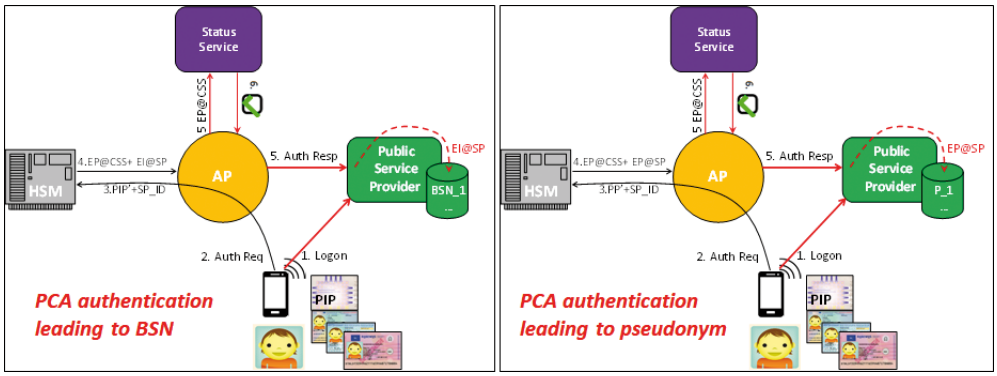

Figure 8. Working of polymorphic card application (PCA)

3 Requirements for the polymorphic eID scheme

In this section we formalize the requirements for the polymorphic eID scheme.As indicated in Section 1.1 we distinguish three types of requirements: reliability, privacy and usability. Our basic assumption is that the Key Management Authority is a trusted third party, i.e. is trusted by all scheme participants in the scheme, i.e. by BSN-L, authentication providers, service providers and users. For simplicity of requirement formulation we do not consider the Key Management Authority a participant of the scheme. parties. We moreover assume that the HSM modules at BSN-K and the authentication providers only allow for the required polymorphic operations.

3.1 eID reliability requirements

Below we specify the reliability requirements for the polymorphic eID scheme.

R_1: Strong authentication The polymorphic eID scheme is consistent with levels Substantial and High of the eIDAS regulation [18,19]. That is, provided authentication providers issue suitably strong authenticators to their clients, the polymorphic eID scheme can support the referred assurance levels.

R_2: PCA design meets eIDAS level High The design of the Polymorphic Card Application (PCA) meets the eIDAS level High requirements [19]. The motivation for this requirement will be primarily based on the properties of the generic eIDAS token described in [5] of which PCA is a simple variant. Also compare requirement $\mathbf{P}_3$ below.

R_3: Independence of identities and pseudonyms The PI/EI and PP/EP structures are cryptographically independent, i.e. the private keys for EI decryption are independent from the private keys for EP decryption. Consequently, the Key Management Authority has technical control over service providers being able to decrypt BSNs and pseudonyms. This requirement will enhance the robustness of the scheme. To indicate, if for instance an authentication provider erroneously sends an EI to a service provider that is only allowed to process pseudonyms, it will not be able to decrypt it.

R_4: BSN-L activation binding It is cryptographically enforced that the authentication provider can only produce *legitimate* Encrypted Identities and/or Encrypted Pseudonyms at service providers, i.e. of existing users, by following the activation protocol through BSN-L. That is, by retrieving the PIP/PI/PP corresponding to the user and transforming it to EI/EP using an HSM. This implies that an authentication provider cannot create usable PIs or PPs himself. Nor can he cannot create such PIs or PPs from those of another authentication provider. This requirement provides technical assurance that the eID user inspection service gives an accurate reflection of activated authentication providers.

R_5: Authenticity of polymorphic forms The integrity and authenticity of polymorphic forms, i.e. PIP, PI, PP, DEP, EI and EP, is cryptographically verifiable by participants. In practice we will endow these forms with electronic signatures where we use a particular signature scheme for the EI and

EP for efficiency reasons. Within PCA, authenticity is based on card secure messaging specified in [5].

3.2 eID privacy requirements

P_1: Indistinguishability of PIs/PPs Different PIs and PPs are indistinguishable for participants, i.e. they cannot assess whether they correspond to the same person. In particular, an authentication provider cannot cryptographically derive the BSN from a PI or a base pseudonym from a PP.

P_2: PCA provides k-anonymity Authentications of a PCA instance cannot be cryptographically distinguished by the authentication provider to a group of holders of size less than a parameter k, cf. [47]. It follows in particular that the authentication provider cannot determine the identity of the holder in a cryptographic fashion. In practice we choose $k = 20.000$ which is motivated in Appendix G. We remark that other information, e.g. IP addresses, browser fingerprints, can allow linking of authentications but this is outside the scope of this document.

P_3: PCA provides privacy friendly revocation PCA supports swift revocation without affecting the pseudonymity of the holder of the card as service providers. The latter requires that as a result of revocation service providers should not be able to link users pseudonym. Compare Appendix F for an issue on this within the German eID card setup.

P_4: Indistinguishability of EIs/EPs Only intended service providers can assess if different EIs (or EPs) correspond to the same person. In particular, only intended service providers can decrypt BSNs from EIs or pseudonyms from EPs.

P_5: Indistinguishability of DEPs Only intended service providers can assess if different DEPs correspond to the same person. In particular, BSN-L cannot assess the pseudonym contained in a DEP.

P_6: Non-invertibility of pseudonyms Service providers are not able to cryptographically derive the identities (BSNs) from their pseudonyms on which they are based.

P_7: Pseudonym unlinkability by SPs Two service providers cannot cryptographically assess if two of their pseudonyms correspond to the same person. It follows that a user is not only provided different pseudonyms at different service providers but that it is cryptographically enforced they cannot link them. Compare Requirement $\mathbf{U}_1$ below.

P_8: Pseudonym unlinkability by SP and AP Given pseudonyms P_1 (respectively P_2) at service provider SP_1 (respectively SP_2). Then SP_1 and an authentication provider cannot cryptographically assess if P_1, P_2 correspond to the same person. That is, the pseudonyms protect against a service provider and an authentication provider colluding. As an authentication provider is considered a trusted role, this requirement can be considered "security in depth". It is primarily meant as further resilience against a compromised authentication provider.

P_9: Conformity of pseudonyms with legal principles Pseudonyms provided to service providers adhere to the principles stipulated in the European General Data Protection Regulation [17] and to the ruling [15] of the Dutch Data Protection Authority of 21 December 2015.

3.3 eID usability requirements

U_1: Pseudonym compatibility The pseudonyms delivered to the service providers are independent of the authentication provider.

U_2: Support for role based pseudonyms The scheme supports that pseudonyms delivered to service providers are not only based on the identities of the user and the service provider but also on a user "role". This, for instance, supports the use case that a user represents another user at a service provider.

4 Cryptographic primitives, notation and conversions

4.1 Background

In this section we specify the cryptographic primitives the polymorphic building blocks (Section 5) is based upon. First of all we let $\mathbb{F}_r$ denote the Galois field consisting of the integers modulo a prime number r. We let $\mathbb{F}_r^*$ denote the multiplicative subgroup, i.e. the non-zero elements. We let $b(r) = \lceil \log_2(r) \rceil$ and $B(r) = \lceil \log_{256}(r) \rceil$ and denote the *size in bits* and *size in bytes* of r respectively. An element x of $\mathbb{F}_r$ can be converted (encoded) to a bit array of length $b(r)$ as

$$x = \sum_{i=0}^{b(r)-1} x_i \cdot 2^i, \tag{1}$$

where each x_i is a bit, i.e. an element of $\{0, 1\}$. An element x of $\mathbb{F}_r$ can also be converted (encoded) to a byte array of length $B(r)$ as

$$x = \sum_{i=0}^{B(r)-1} x_i \cdot 256^i \tag{2}$$

where each x_i is a byte, i.e. an element of $\{0, 1, \ldots, 255\}$. Compare [11, Section 3.1.2]. That is, we allow the most significant bytes to be zero to force the byte array to be of full length $B(r)$. Generally speaking we only care for such full length conversions in the context of field to byte array conversions as here sometimes hash operations are performed for which is relevant. This occurs for instance in digital signatures (cf. Algorithm 1). In those situations standards, e.g. [11], stipulate that zero most significant bytes are also part of the hash operation. We note that this sometimes requires special care of field to byte conversions in implementations.

Central in our construction is an additive group $\mathbb{G} = (\langle G \rangle, +)$ of order q generated by a generator element G. We use additive notation as this is customary in the context of elliptic curve groups we deploy in practice. We assume that q is prime. For any natural scalar n and element $H \in \langle G \rangle$ we define the *(point) multiplication* nH as adding H n-times, e.g. $2H = H + H$. As $nH = mH$ if and only if $n = m \bmod q$ we can represent scalars as elements of $\mathbb{F}_q$. This allows for compact notation as $x \cdot G$, $-x \cdot G$ for $x \in \mathbb{F}_q$ and $y^{-1} \cdot G$ for $y \in \mathbb{F}_q^*$. We sometimes omit the "$\cdot$" symbol and simply write xG. A randomly, or cryptographically secure pseudo randomly, chosen element from a set is denoted by $\in_R$.

The required cryptographic security of the group $(\langle G \rangle, +)$ can be formulated in the intractability of four problems. The first one is the *Diffie-Hellman problem*, which consists of computing the values of the function $DH_G(xG, yG) = xyG$ for any $x, y \in \mathbb{F}_q$ (implicitly given but unknown). We note that the Diffie-Hellman problem is equivalent with inverting the *multiplication trapdoor function*

$$T_x : \langle G \rangle \to \langle G \rangle : H \to x \cdot H,$$

for all $x \in \mathbb{F}_q$ (implicitly given but unknown). This follows as one can calculate xyG on basis of G, xG, yG by inverting $T_{y^{-1}}(xG)$ where $T_{y^{-1}}(.)$ is implicitly given by the condition $T_{y^{-1}}(yG) = G$.

Two other problems are related to the Diffie-Hellman problem. The first one is the *Decision Diffie-Hellman* (DDH) problem with respect to G: given $A, B, C \in_R \langle G \rangle$ decide whether $C = DH_G(A, B)$ or not. The DH problem with respect to G is at least as difficult as the DDH problem with respect to G. The second related problem is the *discrete logarithm* (DL) problem in $\langle G \rangle$ with respect to G: given $A = xG \in \langle G \rangle$, with $x \in \mathbb{F}_q$ then find $x = DL_G(A)$. The DL problem with respect to G is at least as difficult as the DH problem with respect to G.

One can easily show that if one can solve the discrete logarithms with respect to one generator, one can solve it with respect to any generator of $\langle G \rangle$. That is, the hardness of the discrete logarithm problem is independent of the generator of the group. In [51] a similar property is shown for the Diffie-Hellman problem. It seems very unlikely that the hardness of the Decision Diffie-Hellman problem is dependent of the group generator. However, as far as we know such a result is not known to be formally provable. To this end, we say that one can solve the Decision Diffie-Hellman problem with respect to the group G if one can solve the Decision Diffie-Hellman problem with respect to any generator of the group. An equivalent definition is as follows. Any quadruple of points in $\langle G \rangle$ can be written as (H, J, xH, yJ) for some (unknown) $x, y \in \mathbb{F}_q$. The *general Decision Diffie-Hellman problem* amounts to deciding whether a random quadruple of points in G is a *DDH quadruple*, i.e. if $x = y$.

We assume that all four introduced problems in G are intractable which implies that the size $|q|$ of the group order q in bits should be at least 256 bits. Although strictly speaking not necessary, we assume in our constructions that $\mathbb{G}$ is a group of points over a field $\mathbb{F}_p$ on a curve with simplified Weierstrass equation

$$y^2 = x^3 + ax + b \tag{3}$$

for some suitable $a, b \in \mathbb{F}_p$. That is, each non-zero group element takes the form (x, y) where $0 \leq x, y < p$ satisfying Equation (3) modulo p. Compare [25]. We denote the zero element (point at infinity) as $\mathcal{O}$. For practical implementations one can use one of the Brainpool curves [16] or NIST curves [36]. The first version of the Dutch scheme is based on the Brainpool320r1 curve. Here the size in bits of p is 320, i.e. the size in bytes k is 40. The size in bits l of the order q is 320 bit. A non-zero point (x, y) on an elliptic curve is converted to a byte array of length $2k$ by concatenation of x, y each converted to a byte array, i.e. each of size k.

In [11] this conversion is prepended with the byte $0x04$ forming the *uncompressed encoding* of an elliptic curve point. One can also encode a point in *compressed* form consisting of the x-coordinate as a byte array prepended with either the byte $0x02$ or $0x03$ depending of the parity of y. If y is even (respectively odd) corresponds with byte $0x02$ (respectively $0x03$). That is, a point in uncompressed (respectively compressed) encoding consists of $2k+1$ (respectively $k+1$)

bytes. During decompression one first calculates the right hand side of equation 3 using the x-coordinate, solves this for y and picks the one corresponding with the indicated parity. In practical implementations a decompression function is also a convenient way of trying whether a given x corresponds to a point on the curve. See the discussion after Algorithm 11.

4.2 ElGamal encryption

The ElGamal private key y of a user is a random element in $\mathbb{F}_q^*$, whereas the ElGamal public key takes the form $Y = yG$. For $M \in \mathbb{G}$ and $t \in_R \mathbb{F}_q^*$ and $Y = yG$ we let the triple

$$\mathcal{EG}_e(t, M, Y) = \langle\, t \cdot G,\ M + t \cdot Y,\ Y \,\rangle \tag{4}$$

denote the ElGamal encryption [22] of *plaintext* M with respect to the *public key* Y and *private* key y. As can be easily verified, the ElGamal decryption $\mathcal{EG}_d(A, B, C, y)$ of an ElGamal cryptogram $\langle A, B, C\rangle$ is given by:

$$\mathcal{EG}_d(A, B, C, y) = B - yA \tag{5}$$

ElGamal encryption is 'randomised' or 'probabilistic': it uses a random t in each encryption so that encrypting the same message twice gives different ciphertexts with negligible probability of failure. In fact, under the assumption that the Decision Diffie-Hellman (DDH) in $\mathbb{G}$ is intractable it follows that the ElGamal encryption scheme is semantically security, cf. [31, Theorem 10.20]. This roughly says that an adversary cannot decide if two ElGamal encryptions hold the same message.

Strictly speaking the public key Y does not need to be included in the ElGamal encryption $\mathcal{EG}_e$ specification. Indeed, the party for which the encryption is intended does not require it as he already possesses it (or can calculate it from the private key y). We let the public key be part of the ElGamal encryption as it allows for an re-randomisation operation on ElGamal encryptions as we discuss below. See Proposition 4.2. This is a convenient tool to avoid linkability in the e-ID infrastructure.

We can now formalize the three operations Re-Randomization $\mathcal{RR}$, Re-Keying $\mathcal{RK}$ and Re-Shuffling $\mathcal{RS}$ we introduced in Section 2. These operations, naming of which is borrowed from [49], work on triplets holding an ElGamal encryption.

Proposition 4.1 *In the notation introduced above we define three functions $\mathcal{RR}, \mathcal{RK}, \mathcal{RS}$ each with type:*

$$\mathbb{G}^3 \times \mathbb{F}_q^* \longrightarrow \mathbb{G}^3$$

and describe their properties.

1. *The **re-randomisation** of a triple $\langle A, B, C\rangle \in \mathbb{G}^3$ with $r \in \mathbb{F}_q^*$ is defined via the function:*

$$\mathcal{RR}(\langle A, B, C\rangle, r) \stackrel{def}{=} \langle\, r \cdot G + A,\ r \cdot C + B,\ C \,\rangle. \tag{6}$$

If the input is an ElGamal ciphertext, then so is the output:

$$\mathcal{RR}\big(\mathcal{EG}_e(t, M, Y), r\big) = \mathcal{EG}_e(r + t, M, Y). \tag{7}$$

This decrypts to the original message M via the original private key y.

2. *The **re-keying** with $k \in \mathbb{F}_q^*$ is defined via the function:*

$$\mathcal{RK}(\langle A, B, C\rangle, k) \stackrel{def}{=} \langle\, \tfrac{1}{k} \cdot A,\ B,\ k \cdot C\,\rangle, \tag{8}$$

We then have:

$$\mathcal{RK}\big(\mathcal{EG}_e(t, M, Y), k\big) = \mathcal{EG}_e(\tfrac{t}{k}, M, k \cdot Y). \tag{9}$$

This decrypts to the original message M via a different private key $k \cdot y$.

3. *The **re-shuffling** with $s \in \mathbb{F}_q^*$ is defined as a function:*

$$\mathcal{RS}(\langle A, B, C\rangle, s) \stackrel{def}{=} \langle\, s \cdot A,\ s \cdot B,\ C\,\rangle. \tag{10}$$

Then:

$$\mathcal{RS}\big(\mathcal{EG}_e(t, M, Y), s\big) = \mathcal{EG}_e(s \cdot t, s \cdot M, Y). \tag{11}$$

This decrypts to the message sM via the original private key y.

Proof: All results are obtained by easy calculations. As an illustration we prove that equation (7) holds: re-randomisation (6) on an ElGamal encryption yields a new ElGamal encryption of the same message with the same public key, but with random $s + r$, since:

$$\begin{aligned}
\mathcal{RR}\big(\mathcal{EG}_e(r, M, Y), s\big) &\stackrel{(4)}{=} \mathcal{RR}(\langle r \cdot G, r \cdot Y + M, Y\rangle, s) \\
&\stackrel{(6)}{=} \langle s \cdot G + r \cdot G, s \cdot Y + r \cdot Y + M, Y\rangle \\
&= \langle (s + r) \cdot G, (s + r) \cdot Y + M, Y\rangle \\
&= \mathcal{EG}_e(s + r, M, Y). \qquad \square
\end{aligned}$$

The first part of Proposition 4.1 states that one can make different copies of an ElGamal encryption functionally equivalent to the original. The following result states that these copies are not linkable to the original.

Proposition 4.2 *Under the assumption that the Decision Diffie-Hellman is hard in $\mathbb{G}$ the following hold:*

1. *An adversary without knowledge of the private ElGamal key cannot assess if two random ElGamal encryptions under the same public key hold the same message.*
2. *In particular, an adversary cannot link a re-randomized ElGamal encryption $\mathcal{RR}\big(\mathcal{EG}_e(t, M, Y), r\big)$ to the original ElGmal encryption $EG(t, M, Y)$ if r is randomly chosen and unknown by the adversary.*

Proof: See [31, Theorem 10.20]. □

We end this section with a multi-recipient variant of the ElGamal encryption scheme allowing for shortened ciphertexts. This variant is convenient in the situation of smart cards (cf. Section 7) where saving on data storage and communication is essential. In this variant we consider an $n \geq 1$ number of ElGamal public keys $Y_1 = y_1 G, \ldots, Y_n = y_n G$ where all private keys $y_1, \ldots, y_n$ are randomly chosen (and thus also independent). In multi-recipient ElGamal n plaintexts $M_1, \ldots M_n \in \mathbb{G}$ are then encrypted by choosing one $t \in_R \mathbb{F}_q^*$ and forming the $2n + 1$ tuple:

$$\mathcal{MEG}(t, M_1, \ldots, M_n, Y_1, \ldots, Y_n) = \langle\, tG,\ M_1 + tY_1, \ldots,\ M_n + tY_n,\ Y_1, \ldots,\ Y_n \,\rangle.$$

Clearly the map

$$M2S_i(\langle T,\ C_1,\ \ldots,\ C_n,\ Y_1,\ \ldots, Y_n)\rangle) = (T, C_i, Y_i),$$

maps a multi-recipient ElGamal ciphertext to a single-recipient ElGamal ciphertext for the i-th recipient ($1 \leq i \leq n$). This indicates how a recipient of a multi-recipient ElGamal encryption can perform decryption.

One can easily show that the multi-recipient variant of the ElGamal encryption scheme is as secure as the single-recipient one. In fact, this is closely related to the re-keying technique introduced above: based on a single-recipient ElGamal encryption one can simply construct a multi-recipient ElGamal encryption. So the ability to break multi-recipient ElGamal would also allow to break single-recipient ElGamal. Note that it is essential that the private keys are independently chosen; if say two private keys are the same one is obviously leaking plaintext information. See [32,50] for further formalization.

4.3 Digital Signature schemes

In the polymorphic scheme two different digital signatures schemes based on the elliptic curve group $\mathbb{G}$ introduced in Section 4.1. A hash function $H(.)$ shall be used with an output length equal to the bit length l of the group order q. One can use a hash function with an larger output length than l by truncating its output to the l leftmost bits, cf. [11, Section 4.1.2]. The first version of the Dutch scheme is based on the Brainpool320r1 curve and the SHA-384 hash function [35] is used truncated to the 320 leftmost bits. Polymorphic forms (PI, PP, PIP) are signed by the elliptic curve variant of the Digital Signature Algorithm (ECDSA), see [36] and [25, Section 4.4.1]. In our ECDSA context the private key of a user is a random element u in $\mathbb{F}_q^*$, whereas the ECDSA public key takes the form $U = uG$. We denote the creation of an ECDSA signature of a message M with private key u by $Sig = Sig_{\text{dsa}}(M, u)$. The verification of an ECDSA signature Sig on a message M signature with public key U we denote by $Ver_{\text{dsa}}(M, Sig, U)$. The outcome can be True or False.

Encrypted forms (EI and EP) are signed by the elliptic curve variant of the Schnorr signature scheme (EC-Schnorr), see [30] and [**?**]. Like ECDSA also

standard EC-Schnorr is based on a fixed generator G in the elliptic curve group, namely the standard base point G. It is convenient to define EC-Schnorr for any generator J of the elliptic curve group as indicated in the description below. That is, the EC-Schnorr private key d of a user is a random element in $\mathbb{F}_q^*$, whereas the EC-Schnorr public key takes the form of a pair (W, J) where $W = dJ$. With abuse of notation we also call W the EC-Schnorr public key. By taking $J = G$ we arrive at the EC-Schnorr specification of EC-SDSA-opt in [30]. We denote an EC-Schnorr signature of a message M with private key d and generator J, i.e. the pair (r, s) in Algorithm 1, by $Sig_{\text{Schn}}(M, d, J)$.

In our application of EC-Schnorr signatures, the generator J is the scheme Polymorphic Identity public key, i.e. **Y** see Table 6, respectively the scheme Polymorphic Pseudonym public key, i.e. **Z** see Table 6. The EC-Schnorr public keys are the Encrypted Identity public keys of the service providers respectively the Encrypted Pseudonym public keys. This means that the secret value that transforms the scheme public key into a service provider public key plays the role of EC-Schnorr signature key. In general it is not a good idea to give a key two purposes as this might leak information on the key. However, in this situation this is not an issue as EC-Schnorr signatures (unlike ECDSA signatures) are based on a zero-knowledge proof of knowledge. Compare [**?**]. An EC-Schnorr signature is based on the EC-Schnorr interactive proof of knowledge made non-interactive by using the Fiat-Shamir heuristic [23] using a hash value as challenge. The EC-Schnorr interactive proof of knowledge is zero-knowledge. This roughly means that the verifier can not extract secret knowledge from running the protocol as he produce protocol transcripts himself. In other words, EC-Schnorr signatures do not "leak" information on the secret key.

Algorithm 1 $Sig_{\text{Schn}}(M, d, J)$
EC-Schnorr signature creation on message M based on generator J, private key d and public key $Y = dJ$.

1: `Select random` $k \in \{1, ..., q-1\}$.
2: `Compute` $kJ = (x, y)$ `and convert to byte array` $\bar{J}$. `// i.e. of size` $2k$.
3: `Compute` l`-bit bit array` $H(\bar{J}||M)$ `and convert it to an integer` r.
4: `If` $r = 0$ `then go to Line 1.`
5: `Compute` $s = k + r \cdot d \bmod q$.
6: `If` $s = 0$ `then go to Line 1.`
7: `Return` (r, s).

Algorithm 2 $Ver_{\text{Schn}}(M, Sig, Y, J)$
EC-Schnorr verfication of signature $Sig = (r, s)$ on message M based on generator J and public key $Y = dJ$.

1: `Verify that` $r \in \{1, 2^l - 1\}$ `and` $s \in \{1, q - 1\}$`, on failure Return False.`
2: `Compute` $Q = s \cdot J - r \cdot Y$ `if` $Q = \mathcal{O}$ `Return False.`
3: `Convert` Q `to byte array` $\bar{Q}$`.` `// i.e. of size` $2k$
4: `Compute` l`-bit bit array` $H(\bar{Q}||M)$ `and convert it to an integer` v.
5: `If` $v = r$ `Return True otherwise Return False.`

Specifications in Algorithms 1 and 2 are based on the BSI specification of EC-Schnorr in [11, Section 4.2.3]. This coincides with the ISO specification EC-SDSA in [30]. We note that this specification is subtly different from the BSI specification of EC-Schnorr in the 2.0 version of [11] of 2012. The difference is that the 2012 BSI specification is quite similar (but not completely equal) to the optimized version EC-SDSA-opt of [30]. The optimized version does not use $\bar{y}$ in Step 3 of Algorithm 1 and Step 4 of Algorithm 2. In this document we adopt EC-SDSA of [30] despite that the ETSI specification [21] refers to EC-SDSA-opt instead of EC-SDSA. We do not believe that this has impact on the formal "eIDAS qualified signature" status of EC-SDSA especially as the latter is theoretically more secure than EC-SDSA-opt and stricter in line with [23].

4.4 Verifiable ElGamal decryption

In an eID context a user can dispute that a certain authentication at a service provider took place. The service provider then (at least) needs to be able to demonstrate that he received a certain authentication response containing the user identity (BSN or pseudonym). In our eID context, identities take the form of ElGamal cryptograms signed by an authentication provider. Compare Section 5.4. This means that we require the service provider to be able decrypting these cryptograms in a publicly *verifiable* way. That is, that anybody can verify that a certain cryptogram contains a certain identity. In Algorithms 5, 6, 7 and 8 below we specify how this can be achieved. However, it turns out convenient to develop a generic building block for this.

To this end, let $D = d{\cdot}G$ be a public key in $\mathbb{G}$ with corresponding private key $d \in \mathbb{F}_q^*$. Also let $A_1, \ldots, A_n \in \mathbb{G}$. The private key holder forms

$$B_1 = d{\cdot}A_1, \ldots, B_n = d{\cdot}A_n \tag{12}$$

and sends the A_i, B_i to another person (verifier) together with his public key D (implicitly defining d). Now suppose the holder wants to convince the verifier of the form of the B_i in (12). That is, the holder wants to provide the verifier some information T allowing the later to verify this. In fact, this should be publicly verifiable (also known as "transferable"): anybody should be able to verify this. The simplest way to do this would be full disclosure of the holder private key d. However, we require that the information T provided should not leak any secret information on d.

The Schnorr proofs of knowledge [46] allow for this in an interactive protocol between the holder and the verifier. In these proofs the holder first commits to certain values related to (12), the verifier then sends a challenge which the holder can only answer with a suitable response if expression (12) holds. These proofs of knowledge do not "leak" secret information ("zero-knowledge") on d as it can be shown that the verifier essentially does not get any information he could not have generated himself.

The Schnorr protocols can also be made non-interactive (and transferable) using the Fiat-Shamir heuristic [23]. Here the verifier challenge is replaced with a secure hash of the holder commitment. That means that the holder can generate a transcript that allows the verifier (and in fact anybody) to verify that expression (12). We denote such transcript by

$$\mathcal{DT}(A_1, \ldots, A_n \xrightarrow{d} B_1, \ldots, B_n \,|\, D = d{\cdot}G).$$

In the following two algorithms we specify how such transcripts can be created and verified related to expression (12). Their correctness and security follow from [**?**] and [50]. We note that the EC-Schnorr signature scheme from Section 4.3 is also based on these concepts. We also note that the algorithms can easily be extended to cover multiple private keys instead of only one, i.e. d.

Algorithm 3 $\mathcal{DT}_c(A_1, \ldots, A_n \xrightarrow{d} B_1, \ldots, B_n \,|\, D = d{\cdot}G)$
Creation of a transcript by the holder of a private key d.

1: `Select random` $k \in \{1, ..., q-1\}$.
2: `Compute` $k{\cdot}G$, $k{\cdot}A_i$ $(i = 1, \ldots, n)$, `and convert to byte arrays` $\bar{G}, \bar{A}_i$.
3: `Compute` l`-bit bit array` $H(\bar{G}||\bar{A}_1||\ldots||\bar{A}_n)$ `and convert it to integer` r
4: `If` $r = 0$ `then go to Line 1.`
5: `Compute` $s = k + r \cdot d \bmod q$.
6: `If` $s = 0$ `then go to Line 1.`
7: `Return` (r, s).

In Line 2 of Algorithm 3 we write each of the elliptic curve points in their x- and y-coordinates and concatenate these as input to the hash function in Line 3.

Algorithm 4 $\mathcal{DT}_v(A_1, \ldots, A_n, B_1, \ldots, B_n, \mathcal{DT}, D)$
Verification of transcript $\mathcal{DT} = (r, s)$ by a verifier using public key $D = d{\cdot}G$.

1: `Verify that` $A_i, B_i \in \mathbb{G}$ $(i = 1, \ldots, n)$ `on failure Return False.`
2: `Verify that` $r \in \{1, 2^l - 1\}$ `and` $s \in \{1, q-1\}$, `on failure Return False.`
3: `Compute` $Q = s{\cdot}G - r{\cdot}D$, $Q_i = s{\cdot}A_i - r{\cdot}B_i$ $(i = 1, \ldots, n)$
4: `if` $Q = \mathcal{O}$ `or` $Q_i = \mathcal{O}$ $(i = 1, \ldots, n)$ `Return False.`
5: `Convert` Q, Q_i $(i = 1, \ldots, n)$ `to byte arrays` $\bar{Q}, \bar{Q}_i$.
6: `Compute` l`-bit bit array` $H(\bar{Q}||\bar{Q}_1||\ldots||\bar{Q}_n)$ `and convert it to integer` v.
7: `If` $v = r$ `Return True otherwise Return False.`

We remark that by the nature of Schnorr based proofs of knowledge there is a negligible probability (in the order or 2^{-l}, i.e. 2^{-320} in the context of the Brainpool320r1 curve) that Algorithm 4 is erroneously successful. For simplicity we do not further stipitate that in the algorithms. We now can specify how a

private key holder can make it publicly verifiable that an ElGamal cryptogram of his decrypts to a certain plaintext. In Algorithm 5 we specify the creation of decryption transcript by the holder and in Algorithm 6 we specify the public verification thereof.

Algorithm 5 $CDT(\mathcal{EG}, M, y)$
Creation of a decryption transcript proving that $\mathcal{EG} = \langle\, A,\ B,\ Y \,\rangle$ is an ElGamal encryption of $M \in \mathbb{G}$ under public key $Y = y{\cdot}G$.

1: `Return` $\mathcal{DT}_c(A \xrightarrow{y} B - M \,|\, Y = y{\cdot}G)$

Algorithm 6 $VDT(M, \mathcal{EG}, \mathcal{DT})$
Verfication of a decryption transcript $\mathcal{DT} = (r, s)$ proving that an ElGamal encryption $\mathcal{EG} = \langle\, A,\ B,\ Y \,\rangle$ holds M under public key $Y = y{\cdot}G$.

1: `Verify that` $A, B, M \in \mathbb{G}$, `on failure Return False.`
2: `Return` $\mathcal{DT}_v(A, B - M, \mathcal{DT}, Y)$ `// true or false`

Proposition 4.3 *Algorithm 6 applied to Algorithm 5 output is successful if and only if* $\mathcal{EG} = \langle\, A,\ B,\ Y \,\rangle$ *is an ElGamal encryption of* M *under* $Y = y{\cdot}G$.

Proof: Observe that $\mathcal{EG}$ is an encryption of M if and only if

$$\langle A, B - M \rangle = \langle\, t \cdot G,\ t \cdot Y \,\rangle \tag{13}$$

for some $t \in \mathbb{F}_q$. As $Y = y{\cdot}G$ it follows that Formula (13) is equivalent with the condition $B - M = y{\cdot}A$. This condition is conveyed in Algorithms 5 and 6. □

Our eID scheme pseudonyms take the form $P = c{\cdot}M$ where M is the plaintext payload of an ElGamal encryption $\langle\, A,\ B,\ Y \,\rangle$ and the secret multiplication by c ("closing key") is also performed by the private key holder. Compare Section 5.4.3. In this setting it is convenient for the holder to be able to prove this without needing to revealing M. The holder can accomplish this in two steps each using Algorithm 3. He first transforms in a verifiable way the original ElGamal encryption in one holding c-times the plaintext, i.e. allegedly holding $P = c \cdot M$. Then he generates a transcript demonstrating that the new ElGamal encryption holds P by using Algorithm 5. These steps are specified in Algorithms 7 and 8.

Algorithm 7 $CDT(\mathcal{EG}, P, y, c)$
Creation of a decryption transcript that $P = c{\cdot}M$ where the ElGamal encryption $\mathcal{EG} = \langle\, A,\ B,\ Y \,\rangle$ holds M under public key $Y = y{\cdot}G$ and closing key $C = c{\cdot}G$.

1: `Generate` $A' = c{\cdot}A,\ B' = c{\cdot}B$ `// form ElGamal encryption of` P
2: `Form` $\mathcal{DT}_1 = \mathcal{DT}_c(A, B \xrightarrow{c} A', B' \,|\, C = cG)$
3: `Form` $\mathcal{DT}_2 = \mathcal{DT}_c(A' \xrightarrow{y} B' - P \,|\, Y = y{\cdot}G)$
4: `Return` $A', B', \mathcal{DT}_1, \mathcal{DT}_2$

Algorithm 8 $VDT(\mathcal{EG}, P, y, c)$
Verification of a decryption transcript $U, V, \mathcal{DT}_1, \mathcal{DT}_2$ that $P = c \cdot M$ where the ElGamal encryption $\mathcal{EG} = \langle\, A,\ B,\ Y \,\rangle$ holds M under public key $Y = y{\cdot}G$ and closing key $C = c{\cdot}G$.

1: `Verify that` $A, B \in \mathbb{G}$ `on failure Return False.`
2: `If` $\mathcal{DT}_v(A, B, U, V, \mathcal{DT}_1, C)$ `if False return False`
3: `If` $\mathcal{DT}_v(U, V - P, \mathcal{DT}_2, Y)$ `if False return False`
4: `Else return True`

Proposition 4.4 *Algorithm 8 applied to the output to Algorithm 7 is successful if and only if $P = c{\cdot}M$ where $\mathcal{EG} = \langle\, A,\ B,\ Y \,\rangle$ is an ElGamal encryption of M under public key $Y = y{\cdot}G$ and closing key $C = c{\cdot}G$*

Proof: If M is the plaintext payload of $\mathcal{EG}$ then Line 1 of Algorithm 7 generates an ElGamal encryption of $c{\cdot}M$ by the third part of Proposition 4.1. Line 2 of Algorithm 7 makes this publicly verifiable which is verified in Line 2 of Algorithm 8. Similar to Proposition 4.4 it now follows that $\mathcal{DT}_v(U, V - P, \mathcal{DT}_2, Y)$ is True if and only if the payload contents of the ElGamal encryption $\langle\, A,\ B,\ Y \,\rangle$ equals $P = c{\cdot}M$. □

4.5 Symmetric encryption (AES-256)

The polymorphic scheme uses symmetric encryption to protect the confidentiality of a so-called 16 byte audit block. Compare Sections 5.2.1, 5.2.2, 5.2.3. For this we deploy the 256 bit variant of the Advanced Encryption Standard (AES) [34] in its simplest mode namely Electronic Code Book (ECB). In this mode one uses a 256 bit key K to encrypt a 16 byte plaintext block M resulting in ciphertext block C also of size 16 byte. We denote this as $C = \mathcal{E}_{\text{AES}}(M, K)$.

4.6 Key derivation functions

The polymorphic scheme is heavily based on key derivation functions $\mathcal{K}(.,.)$. These functions enable to derive secret cryptographic key $\mathcal{K}(K, D)$ from a master key K and an arbitrarily sized *derivation byte array* D. In our applications of derivation functions the derivation byte array D will typically be based on a string S. With slight abuse of notation we also allow notation like $\mathcal{K}(K, S)$. Here we implicity interpret the string S by its canonical byte array representation (without trailing zero byte). In case of a regular string, i.e. consisting of printable ASCII characters, this thus consists of its ASCII byte values in the range 0x20-0x7E.

The basic KDF security property is that, provided the master key K is suitable chosen, derived keys are practically as secure as truly random keys. In the polymorphic scheme we require three key derivation functions $\mathcal{K}_1(.,.)$, $\mathcal{K}_2(.,.)$, $\mathcal{K}_3(.,.)$. Here $\mathcal{K}_1(.,.)$ takes its values in $\mathbb{F}_q^*$ and $\mathcal{K}_2(.,.)$ takes its values in $\mathbb{F}_p^*$. That is, both these derivation functions lead to non-zero values. The derivation

function $\mathcal{K}_3(.,.)$ takes its values in $\{0,1,\ldots,255\}^{32}$, i.e. as byte arrays of length 32 These values corresponds to AES-256 keys, cf. Section 4.5. $\mathcal{K}_1 : (K, D) \to \mathbb{F}_p^*$ and $\mathcal{K}_2(.,.) \to \mathbb{F}_q^*$. That is, in both cases, derived keys should never be zero. For security assurance and ease of implementation we base the key derivation functions on standardized algorithms: they are based on the recommendations [40] of the US National Institute of Standards and Technology (NIST) combined with the recommendations of the German Bundesamt für Sicherheit in der Informationstechnik (BSI) [11].

To this end, we deploy the KDF in Counter Mode from [40, Section 5.1] using the HMAC-SHA384 function [37,35] as pseudorandom function. We further choose to let the Context string be equal to D. We do not use a Label string, i.e. Label is the empty string. We do use the zero byte separating the Label from the output size, i.e. $[L]_2$, in the construction. Also we let choose the length r of the binary representation of the counter to be the smallest possible multiple of 8. The input data encoding will be byte oriented: every input needs to formatted into the smallest byte array fitting. Now let $\mathcal{K}_N(K, D, L)$ denote the resulting pseudorandom key of size L bits based on master key K and derivation byte array D. As before, let k be the size in bytes of the group order q. In line with the approach of [11, Section 4.1.1] we now define:

$$\mathcal{K}_1(K, D) = 1 + (\mathcal{K}_N(K, D, 8k + 64) \bmod (q - 1)). \tag{14}$$

Here we add one to a pseudorandom element modulo $q - 1$ where 64 more bits are generated than the length k of $q - 1$. If we would generate precisely k bits a small bias for small numbers would arise. By generating 64 more bits we still do not produce a formally uniform distribution modulo $q - 1$ but the deviation from it is assumed not to be exploitable. Compare [11]. By adding one to a pseudorandom element modulo $q-1$ ensures the output is always a non element in $\mathbb{F}_q$. Like wise we let

$$\mathcal{K}_2(K, D) = 1 + (\mathcal{K}_N(K, D, 8l + 64) \bmod (p - 1)), \tag{15}$$

where l is the size of p in bytes. We also require a key derivation function for 256 bit AES keys, i.e. leading to 32 byte arrays. For this we define:

$$\mathcal{K}_3(K, D) = \mathcal{K}_N(K, D, 32) \tag{16}$$

In the situation of the Brainpool320r1 curve we have $k = l = 40$. Moreover, as the output size of SHA-384 is 384 bits we only need one iteration of the KDF of [40]. It follows that (letting $\texttt{HMAC-SHA384}_{32}$ denote the truncation to the first, left, 32 bytes):

$$\left.\begin{aligned} \mathcal{K}_1(K, D) &= 1 + (\texttt{HMAC-SHA384}(K, 0x01 \,||\, D \,||\, 0x0180) \bmod (q - 1)) \\ \mathcal{K}_2(K, D) &= 1 + (\texttt{HMAC-SHA384}(K, 0x01 \,||\, D \,||\, 0x0180) \bmod (p - 1)) \\ \mathcal{K}_3(K, D) &= \texttt{HMAC-SHA384}_{32}(K, 0x01 \,||\, D \,||\, 0x0180). \end{aligned}\right\} \tag{17}$$

The term "0x01" indicates the counter value of the first block as we only require one such block. The term "0x0180" represents the length of the output in bits

in hexadecimal form, i.e. 384 in decimal notation. We note that as we use the KDF in Counter Mode from [40, Section 5.1] with only one block, it actually coincides with the KDF in Feedback Mode from [40, Section 5.2] whereby using the optional iteration counter i that is compulsory in counter mode.

4.7 Encoding identities as group elements

For the construction of the polymorphic identity and encrypted identity we require that user identifiers are encoded as elements of the group $\langle G \rangle$, i.e. as an elliptic curve point. That is, based on the point one can determine the user identifier. This implies that the size of user identifiers is limited by the data that can be stored in an elliptic curve point. To this end, within each version of a polymorphic scheme we distinguish the parameter *Identifier Length*, an integer m: we assume that all user identifiers can be represented as a byte array of length m. As we shall see m is upper bounded by the *EC Field Length*, i.e. the length in bytes k of the prime number p over which the elliptic curve group size is defined. We also distinguish an *OAEP hash length* a positive integer h. Different versions of the polymorphic scheme can have different choices for m, k and h. In the first polymorphic version we have $k = 40$ (the prime number is 320 bits) and, as indicated below, $m = 18$ and $h = 10$.

The embedding of user identifiers is based on Optimal Asymmetric Encryption Padding (OAEP) [3]. We refer to the OAEP encoded message EM as specified in Section 7.1 of the PKCS #1 standard [28]. with the specific choices/-parameters indicated in Configuration 4.5.

Configuration 4.5 *Let $\mathcal{H}(.)$ be a secure hash function and let $\mathcal{H}_h(.)$ denote its truncation to the h leftmost bytes. With respect to OAEP encoding [3] we deploy the following specific choices/parameters:*

- *We assume that identifiers can be represented as byte arrays of size m.*
- *the role of n (RSA modulus) in [3] is taken by the prime p defining the field defining the elliptic curve. The length in bytes of p is denoted by k.*
- *We assume that $k = m + 2h + 2$.*
- *The hash function deployed will be $\mathcal{H}_h(.)$ where as Mask Generation Function will be MGF1 defined in [28] using the (full) function $\mathcal{H}(.)$.*
- *the optional Label is the empty string.*

In this context, [3] defines an encoding of an identifier of length m bytes into k bytes, where the first byte is 0x00. That is, the encoding can be considered as an element of $\mathbb{F}_p$.

In regular OAEP encoding the hashlength h is taken equal to the output length of the hash function used. This typically is not an issue as RSA moduli are typically very large, e.g. 2048 bits. We apply OAEP in the context of the elliptic curves where moduli are no so large. The first version of the Dutch eID scheme is based on Brainpool320r1 curve and the SHA-384 hash function whose output length is too large. This is why we have truncated the output of the hash function

to h bytes. The prime number defining the field $\mathbb{F}_p$ used in the Brainpool320r1 curve is of length 320 bits, i.e. 40 bytes. That is, we in the first Dutch scheme we have $k = 40$. As we will explain in Section 8, the choice of h is a balance between "plain-text awareness" of the polymorphic forms and the size m the user identifier can take. In the first version of the Dutch scheme we take $m = 18$ and $h = 10$.

The embedding of an identity Id of type T as a point in $\mathbb{G}$ is specified in the pseudo-code of Algorithm 9. Algorithm 10 specifies the decoding operation. The function $E(.)$ representing Id as a byte array in Line 1 of Algorithm 9 and corresponding decoding function $D(.)$ in Line 5 of Algorithm 10 are specified in Appendix B. With the choices $k = 40$, $m = 18$ and $h = 10$ the encoding function $E(.)$ and thus Algorithm 10 supports identities of size at most 15 ASCII characters. We note that Algorithm 9 is similar to Algorithm 11.

Algorithm 9 $\mathcal{EMB}(Id, m, k, h)$
OAEP encoding of identity string Id of type T as point P in group $\mathbb{G}$.

1: `Form byte array` $S = E(Id, T, m)$ `on failure return Error` `// Appendix B`
2: **repeat**
3: `Form OAEP encoded message EM of` S `using Configuration 4.5.`
4: `Consider EM as an element of` $f \in \mathbb{F}_p$.
5: **until** `A point` P `in` $\mathbb{G}$ `exists with` f `as x-coordinate.`
6: `Return point` P `in` $\mathbb{G}$ `with` f `as x-coordinate and y-coordinate even.`

Algorithm 10 $\mathcal{DEC}(P, m, k, h)$
OAEP decoding of point P in group $\mathbb{G}$ to identity string Id and type T .

1: `Validate if the y-coordinate of` P `is even, otherwise return Error`
2: `Consider the x-coordinate` X `of` P `as a byte array of size` k.
3: `Let` O `be the OAEP decoding of` X `using Configuration 4.5.`
4: `If OAEP decoding fails, return an error`
5: `Return` $(Id, T) = D(O, m)$ `// Appendix B`

In Line 5 of Algorithm 9 one fills in value f in the equation defining the curve, i.e. Equation (3), and tries to solve this for y. That is, one validates if the righthand side of Equation 3 is a quadratic residue in $\mathbb{F}_p$. The probability that is the case is half. In this case precisely two solutions exist, namely of the form y and $p - y$ with $0 \leq y < p$. As p is odd, exactly one of the two y-coordinates must be even. This one is returned by Algorithm 9. We note that in practice one often does not use the full point (x, y) to represent a point on an elliptic curve but only value x plus a parity indication of y, i.e. if it is even or odd. This "compression" is formalized in various standards, e.g. [11]. Here an even y is indicated with an x prefix 0x02 and an odd y is indicated with prefix 0x03. The uncompressed representation consists of the concatenation of values x and y prepended with prefix 0x04. A "decompression" routine in a cryptographic library used can conveniently support Line of Algorithm 9. One simply prepends 0x02 to the octal representation of value f and tries to decompress the result.

If successful the condition in Line 5 is satisfied and the uncompressed point represents the required point.

4.8 Keyed mapping of identities to group elements

Algorithm 11 below specifies how a string Id holding a user identity and the identity type T is mapped into the group $\langle G \rangle$. This algorithm is similar to Algorithm 9. The mapping $I(.)$ in Line 1 of Algorithm 11 is specified in Appendix B. The function I2OS(.) in Algorithm 11 converts an non-negative integer i into an octal string pertaining its minimal representation to the base 256 most significant bytes first.

Algorithm 11 $\mathcal{W}(K, Id)$
Keyed mapping of identity string Id as point P in group $\mathbb{G}$

1: `Set` $i = 0$
2: **repeat**
3: $\quad f = \mathcal{K}_2(K, I(Id, T) \ \| \ \texttt{I2OS}(i))$
4: $\quad i = i + 1$
5: **until** `A point` P `in` $\mathbb{G}$ `exists with` f `as x-coordinate.`
6: `Return point` P `in` $\mathbb{G}$ `with` f `as x-coordinate and y-coordinate even.`

In Line 5 of Algorithm 11 one fills in value f in the equation defining the curve, i.e. Equation (3), and tries to solve this for y. That is, one validates if the righthand side of Equation (3) is a quadratic residue in $\mathbb{F}_p$. The probability that is the case is half. In this case precisely two solutions exist, namely of the form y and $p - y$ with $0 \leq y < p$. As p is odd, exactly one of the two y-coordinates must be even. This one is returned by Algorithm 11. As explained at the end of Section 4.7 one can conveniently use the "decompression" routine in a cryptographic library to implement Line 5 of Algorithm 11.

4.9 Keys, cryptograms and versioning

For controlled change of keys (roll-over) it is required formalizing the notions keys and cryptograms used in this document. We start with formalizing keys in Section 4.9.1. Cryptograms are formalized in Section 4.9.2. In Section 4.9.3 we discuss the usage of time stamps in keys and cryptograms.

4.9.1 Keys and versioning

A cryptographic *key* is the output of a cryptographic operation either based on a randomness source or deterministically based on other cryptographic keys and derivation data. In the first case a key is called *non-derived* and in the latter case is called *derived*. In our model we do not consider keys that are formed on both a randomness source and other cryptographic keys and derivation data. Table 6 in Appendix C indicates all non-derived and derived keys specified in this paper.

We formalize the key structure in Java-pseudo code allowing to stay close to implementations.

```
class Key {
int SchemeVersion
string Creator
string Recipient
string GenerationTime
string ActivationTime
int KeyType
int KId
int[] KVS // KV Sequence
          //i-th entry non-zero
          //iff i-th key is used
byte[][] Keyd // key data
}
```

As indicated, a key structure consist of *key metadata* and the actual *key data*. The key metadata consists of the *fields* SchemeVersion, Creator, Recipient, GenerationTime, ActivationTime, KeyType, KId (Key Identifier) and KVS. The Keyd array of byte arrays in a key contain the actual key data (or handles to it). As in Java we assume that the length of both KVS and Keyd can be derived from it. The actual form of Keyd depends on the implementation and the key type which is indicated by KeyId and SchemeVersion.

We have also explicitly included a KeyType. In the present document the KeyType can indicate one of more of the four types indicated in Table 6 in Appendix C.1): EC private key, EC public key, HMAC key and AES key. The KeyType is superfluous as the key type will also follow from the KeyId and SchemeVersion; it is only included for robustness reasons. The KeyType field is bit-field where the i-th least significant bit indicates the key of type of the key. We let 0x01 indicate an EC Private Key, 0x02 an EC Public Key, 0x04 an HMAC key and 0x08 an AES Key. This allows the Keyd array holding several key types simultaneously. With the exception of four keys (namely $\mathbf{DT_{Di,R}}$, $\mathbf{DT_{Di,R}}$, $\mathbf{PS_{Di,R}}$, $\mathbf{DR_{Di,R}}$ see Table 6 in Appendix C.1), Keyd consists of only one byte array (holding only one key type).

As also indicated, each key has a SchemeVersion, which is one (1) for the first scheme, and a creator and recipient of the key. A key also has its own key type identifier (KId) which is a number $i = 1, 2, \ldots, n$ where n is the (maximum) number of key types in a certain SchemeVersion. As indicated in Table 6 the present scheme contains 27 keys, i.e. $n = 27$, and the key type identifier is denoted in the second column.

A key also has a field indicating its generation and activation times. The activation time is typically later than the generation time allowing early communication to parties that the key is going to be activated. See Section 10. We note that the generation and activation times should be rather precise, see Section 4.9.3.

The Key Version Sequence (KVS) holds all the key identifiers and key versions on which a key is based; a key with type identifier (KId) is used if and only if the Kid entry in KVS is non-zero in which case it holds the key version. By construction, the KVS of the key's own type identifer is non-zero, holding the key's own version. We let corresponding public, private keys have the same key version. We note that a non-derived key only has a non-zero KVS entry for its own key identifier. The non-zero KVS entries of a derived key indicate the keys from which the key is derived from as well as the key versions used therein. Derived keys are only derived from non-derived keys. That is, the KVS of a derived key only refers to non-derived keys and to itself. For a derived key we require that the key data (Keyd) can be completely reconstructed from all its key fields and keys indicated in the KVS. If a derived key uses other derivation data than included as standard fields, then these fields needs to be added to the key metadata. The KeyId will allow processing applications anticipating these fields. We note that keys $\mathbf{DT_{Di,R}}, \mathbf{DT_{Di,R}}, \mathbf{PS_{Di,R}}, \mathbf{DR_{Di,R}}$ can have an optional "Role" field and keys $\mathbf{SED_a}, \mathbf{SED_t}$ have an "Auditee" role. Compare Table 6 in Appendix C.1). Appendix C.2 specifies how the derived keys in this document are formed.

As in Java we use "." to select the member in the above classes. So for instance if K is a key then K.Creator and K.KVS denote the creator string and the key version sequence of K respectively. Two keys K_1, K_2 are called *compatible* if they belong to the same scheme version and are based on the same key versions. That is, if for certain $1 \leq i \leq n$ both K_1.KVS$[i]$ and K_2.KVS$[i]$ are non-zero then they are equal, i.e. hold the same key version.

A particular situation in which keys are compatible occurs when they are *independent*, i.e. when they belong to same scheme version but are not based on any common keys. That is, independent keys do not have KVS entries in common that are both non-zero.

The Keyd array of byte arrays in a key contains the actual key data (or handles to it). The actual form depends on the implementation and the key type indicated by KeyId and SchemeVersion. With the exception of two keys (namely $\mathbf{DT_{Di,R}}$, $\mathbf{DT_{Di,R}}$ see Table 6 in Appendix C.1), Keyd consists of only one byte array. The key indicator together with the SchemeVersion allows determination of the cryptographic operation in which Keyd is formed.

4.9.2 Cryptograms and versioning

The notion of a *cryptogram* is similar that of a key. A cryptogram is the output of a cryptographic operation based on (derived) cryptographic keys, user data and other cryptograms. We require that a cryptogram contains a scheme version (SchemeVersion) which has the same meaning as in key structures. A cryptogram also contains a cryptogram identifier CrID which has meaning within the scheme version. The combination of the scheme version and a cryptogram identifier allows the determination of the cryptographic operation in which the cryptogram

is formed. We define a cryptogram structure in Java-pseudo code allowing to stay close to implementations.

```
class Cryptogram {
    int SchemeVersion
    int CrId
    string Creator
    string Recipient
    string GenerationTime
    int[] KVS // KV Sequence
              //i-th entry filled
              //iff i-th key is used
    byte[][] CrData
}
```

Similar to key structures, a cryptogram consists of metadata (i.e. all the fields) and cryptogram data (i.e. CrData). We require that the scheme version of a cryptogram coincides with that in all the key indicators in its key indicator array. The CrData array of byte arrays contains the cryptogram data of which the construction and form depends of the cryptogram type indicated by CrId and SchemeVersion. If a cryptogram type requires additional metadata for its processing, then these need to be added to the cryptogram metadata. The KeyId will allow processing applications anticipating these fields. As an illustration, an EP and DEP hold an optional role allowing different pseudonyms within a service providers. Also, DEP/DEI cf. Section 16, 15, hold the party authorized to use these. The KVS of the newly formed cryptogram reflects the keys and key versions it is based upon. In principle we only allow cryptograms formed on cryptograms and keys that are compatible. The key roll-over techniques discussed in Section 10 form an exception to this principle. Here we will allow processing cryptograms and keys corresponding with different key versions.

As indicated, cryptograms have included a generation time in a cryptogram. We note that the generation time in a cryptogram should not be too precise to avoid linkability issues. See Section 4.9.3.

If X represents this structure, then X.SchemeVersion indicates the scheme version on which the cryptogram is based et cetera.

4.9.3 Timestamps

The cryptographic key description introduced include time stamps relating to its generation and activation date/time. Likewise the cryptogram description introduced includes a time stamp relating to its generation date/time. The generation and activation times indicated in keys needs to be a rather precise, we suggest taking the time in seconds since 1 January 1970 UTC. To avoid linkability issues the generation time in cryptograms should not be very precise. To indicate, if one observes when a user is issued a polymorphic form then a precise generation time inside the form allows linking the form to the user. To address this

we suggest taking the time in the form YYYYMM (year-month). This allows relying parties to assess that a polymorphic form is not "too old" without introducing such linkability issues. We remark that all polymorphic form contain a so-called audit block for the scheme supervisor that also includes a time stamp. This time stamp consists of the time in seconds since 1 January 1970 UTC, i.e. is rather precise. As the audit block contents is only decipherable by the scheme supervisor this does not introduce linkability issues.

5 Polymorphic cryptographic building blocks

5.1 Introduction

In this section we specify the cryptographic building blocks functionally introduced in Section 2 and most notably Section 2.2. These specifications are based on the cryptographic primitives specified in Section 4. For the convenience of the reader we we have documented all terms and abbreviations used in these specifications in Appendix A. This appendix also contains particular choices made in the first version of the polymorphic scheme. The cryptographic keys referred to in the specifications are documented in Appendix C.

In the specifications we implicitly assume that cryptographic forms are represented in a structured message format, e.g. XML [52], JSON [27] or a TLV structure [29]. This means that we assume that a for instance a triple of elliptic curve points is represented in a form unambiguously clear for a relying party, including the parameters of the curve. If cryptographic forms are digitally signed we also assume that the representation information is also signed. We will not further elaborate on this in the specifications, but for readability we sometimes explicitly introduce a form identifier. The table below outlines which cryptographic algorithms are specified in which sections below.

Section	Cryptographic algorithms specified
5.2	Generation of: • Polymorphic Identities (PIs), • Polymorphic Pseudonyms (PPs), • Polymorphic Identities and Pseudonyms (PIPs), • Direct Encrypted Pseudonyms (DEPs). • Direct Encrypted Identities (DEIs).
5.3	Generation of: • Encrypted Identities (EIs), • Encrypted Pseudonyms (EPs).
5.4	Validation and decryption of: • Encrypted Identities (EIs), • Encrypted Pseudonyms (EPs), • Direct Encrypted Pseudonyms (DEPs). • Direct Encrypted Identities (DEIs).

All polymorphic forms are equipped with an *audit block*. This is a structure consisting of 16 byte holding an AES encryption of the concatenation of a 4 byte HSM identifier (HSM_{ID}), a 4 byte time indicator (e.g. holding seconds since 1 January 1970 UTC) and a 8 byte serial number (SN).

5.2 Generation of polymorphic forms at BSN-L during activation

As part of its activation service, BSN-L maintains Hardware Security Modules (HSMs) to generate polymorphic forms. As indicated in Section 5.1 each HSM in the scheme is identified with a unique HSM_{ID} of 4 byte administered by the KMA. Moreover each HSM is able to produce a timestamp T (4 byte) and a sequence number SN (8 byte) starting with zero. For each polymorphic form returned the HSM increments the sequence number. As indicated in Table 6 of Appendix C.1, BSN-L is provided the following keys by the KMA:

- $\mathbf{Y}$: an ElGamal public key called Identity Private Public key
- $\mathbf{Z}$: an ElGamal public key called Pseudonym Private Public key
- $\mathbf{IW_M}$: an HMAC master key called Identity Wrapping Master key
- $\mathbf{IM_M}$: an HMAC master key called Identity Mapping Master key
- $\mathbf{AA_M}$: an HMAC master key called Authentication provider Adherence Master key. The master key $\mathbf{AA_M}$ is used by BSN-L to derive for each authentication provider a key $\mathbf{AA_{Di}}$ called Authentication provider Adherence Derived key. The $\mathbf{AA_{Di}}$ key ensures that PI/PP/PIPs are authentication provider specific. From the master key $\mathbf{AA_M}$ BSN-L also derives the $\mathbf{SED_a}$ keys used to encrypt data for the supervisor inside the PI/PP/PIPs.
- $\mathbf{DT_{Di,R}}$: keys related to the construction of Direct Encrypted Pseudonyms. This key type consists of two parts, an EC private key and an ElGamal public key. BSN-L gets such a key for each service provider it generates Direct Encrypted Pseudonyms for (optionally relating to role R), most notably the user inspection service. The public key in $\mathbf{DT_{Di,R}}$ is independent of the optional role R.
- $\mathbf{ID_{Pi}}$: an ElGamal public key related to the construction of Direct Encrypted Identities. BSN-L gets such a key for each service provider it generates Direct Encrypted Identities for.

Next to the keys provided by the KMA, BSN-L generates its own signing public/private key pair $\mathbf{U}, \mathbf{u}$ of type ECDSA.

BSN-L processing rules
In all algorithms below, BSN-L will always use the most recent version of the keys it possesses which will be reflected in the KeyIndicator of the resulting cryptogram. See Section 4.9.

5.2.1 Generation of Polymorphic Identity (PI)

The algorithm below specifies how a Polymorphic Identity is generated for a string "identity". In implementations the actual string that is embedded will typically be augmented with identity metadata, e.g. with version and type indicators. We let $\mathrm{PI_M}, \mathrm{uPI_M}$ denote the cryptogram metadata associated with a (unsigned) Polymorphic Identity. In line with Section 4.9, both metadata contain the authentication provider for which the (unsigned) Polymorphic Identity is meant.

Algorithm 12 *GenPI*$(\mathrm{AP_{ID}}, Id)$
Generate Polymorphic Identity for $\mathrm{AP_{ID}}$ based on *Id*.

1: $SN{=}SN{+}1$ // increment sequence number.
2: Compute $P_1 = \mathcal{EMB}(Id, m, k, h)$ // OAEP embed Id into curve
3: Generate key $\mathbf{AA_{Di}}$ // see Appendix C.2
4: Compute $P_2 = \mathbf{AA_{Di}}^{-1} \cdot P_1$ // make AP specific
5: Generate $t \in_R \mathbb{F}_q^*$ and compute $E = \mathcal{EG}_e(t, P_2, \mathbf{Y})$ // form base PI
6: $\mathtt{uPI} = (\mathtt{uPI_M}, E)$ // form unsigned PI
7: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
8: Generate key $\mathbf{SED_a}$ // see Appendix C.2
9: $AB_2 = \mathcal{E}_{\mathrm{AES}}(AB_1, \mathbf{SED_a})$ // encrypt AB for Supervisor
10: Represent $\mathtt{uPI}, AB_2, \tilde{T}, \mathbf{u}.\mathtt{KV}$ as byte array B // e.g. DER encoding
11: Compute $Sig = Sig_{\mathrm{dsa}}(B, \mathbf{u})$ // sign
12: Return $\mathtt{PI} = (\mathtt{PI_M}, \mathtt{uPI}, AB_2, \tilde{T}, Sig)$

5.2.2 Generation of Polymorphic Pseudonym (PP)

We let $\mathrm{PP_M}, \mathrm{uPP_M}$ denote the cryptogram metadata associated with a (unsigned) Polymorphic Pseudonym.

Algorithm 13 *GenPP*$(\mathrm{AP_{ID}}, Id)$
Generate Polymorphic Pseudonym for $\mathrm{AP_{ID}}$ based on *Id*.

1: $SN{=}SN{+}1$ // increment sequence number.
2: Compute $Q_1 = \mathcal{W}(\mathbf{IW_M}, Id)$ // keyed mapping of Id into curve
3: Compute $Q_2 = \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot Q_1$ // additional keyed mapping in curve
4: Generate key $\mathbf{AA_{Di}}$ // see Appendix C.2
5: Compute $Q_3 = \mathbf{AA_{Di}}^{-1} \cdot Q_2$ // make AP specific
6: Generate $t \in_R \mathbb{F}_q^*$ and compute $E = \mathcal{EG}_e(t, Q_3, \mathbf{Z})$ // form base PP
7: $\mathtt{uPP} = (\mathtt{uPP_M}, E)$ // form unsigned PP
8: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
9: Generate key $\mathbf{SED_a}$ // see Appendix C.2
10: $AB_2 = \mathcal{E}_{\mathrm{AES}}(AB_1, \mathbf{SED_a})$ // encrypt AB for Supervisor
11: Represent $\mathtt{uPP}, AB_2, \tilde{T}, \mathbf{u}.\mathtt{KV}$ as byte array B // e.g. DER encoding
12: Compute $Sig = Sig_{\mathrm{dsa}}(B, \mathbf{u})$ // sign
13: Return $\mathtt{PP} = (\mathtt{PP_M}, \mathtt{uPP}, AB_2, \tilde{T}, Sig)$

5.2.3 Generation of Polymorphic Identity and Pseudonym (PIP)

We let $\mathrm{PIP}_{\mathrm{M}}, \mathrm{uPIP}_{\mathrm{M}}$ denote the cryptogram metadata associated with a (unsigned) Polymorphic Identity and Pseudonym.

Algorithm 14 *GenPIP*($\mathrm{AP}_{\mathrm{ID}}, Id$)
Generate Polymorphic Identity Pseudonym for $\mathrm{AP}_{\mathrm{ID}}$ based on *Id*.

1: SN=SN+1 // increment sequence number.
2: Compute $P_1 = \mathcal{EMB}(Id, m, k, h)$ // OAEP embed Id into curve
3: Generate key $\mathbf{AA_{Di}}$ // see Appendix C.2
4: Compute $P_2 = \mathbf{AA_{Di}} \cdot P_1$ // make AP specific
5: Compute $Q_1 = \mathcal{W}(\mathbf{IW_M}, Id)$ // keyed mapping of Id into curve
6: Compute $Q_2 = \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot Q_1$ // additional keyed mapping in curve
7: Compute $Q_3 = \mathbf{AA_{Di}} \cdot Q_2$ // make AP specific
8: Generate $t \in_R \mathbb{F}_q^*$ and compute $E = \mathcal{MEG}(t, P_2, Q_3, \mathbf{Y}, \mathbf{Z})$ // form base PIP
9: $\mathtt{uPIP} = (\mathtt{uPIP_M}, E)$ // form unsigned PIP
10: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
11: Generate key $\mathbf{SED_a}$ // see Appendix C.2
12: $AB_2 = \mathcal{E}_{\mathtt{AES}}(AB_1, \mathbf{SED_a})$ // encrypt AB for Supervisor
13: Represent $\mathtt{uPIP}, AB_2, \tilde{T}, \mathbf{u}$.KV as byte array B // e.g. DER encoding
14: Compute $Sig = Sig_{\mathtt{dsa}}(B, \mathbf{u})$ // sign
15: Return $\mathtt{PIP} = (\mathtt{PIP_M}, \mathtt{uPIP}, AB_2, \tilde{T}, Sig)$

5.2.4 Generation of Direct Encrypted Identity (DEI)

For completeness reasons, and not used in the protocols described in this document, we also introduce a special form of an encrypted identity called *Direct Encrypted Identity* or DEI. It contains the user identity (BSN) in a form decipherable for the intended service provider. A DEI is similar to that of an encrypted identity (EI) but avoids the polymorphic transformation step which can be required in certain use case. See Section 5.3.1. As in the situation of a DEP two parties are involved in a DEP next to BSN-L: the party requesting it and the intended service provider to which the DEP is finally sent and that can decrypt it.

We let $\mathrm{DEI}_{\mathrm{M}}, \mathrm{uDEI}_{\mathrm{M}}$ denote the cryptogram meta associated with a Direct Encrypted Identity. Similar to a Direct Encrypted Pseudonym, the intended service provider is denoted in the Recipient metadata field and the party requesting the DEI is reflected in an additional field in the metadata.

A particular use case arises when the receiving party is an authentication provider and BSN-L itself is the intended service provider. This use case then conveniently allows for "recurrent activation", a periodic recurrence of the activation process by an authentication provider providing assurance on the user status. Recall that the activation process requires the user identity (BSN) which the AP is obliged to delete after activation. This can be addressed by providing the AP a DEI as part of activation. Alternatively one can require an AP to periodically perform the full activation process.

Usage of the DEI service is subject to an application process deciding if the service is allowed for a particular requester/service provider. If successful, the application process will lead to the issuance to BSN-L of an active public key $\mathbf{ID_{Pi}}$ related to encrypted identities and the corresponding private key $\mathbf{ID_{Di}}$ to the intended service provider.

We remark that DEIs can in principle be generated with any ElGamal public/private key pair and not only those of type $\mathbf{ID_{Pi}}, \mathbf{ID_{Di}}$. However, using the latter is more efficient.

Algorithm 15 *GenDEI*(*Id*)
Generate Direct Encrypted Identity for BSN-L based on *Id*.

1: $SN{=}SN{+}1$ // increment sequence number.
2: Compute $P_1 = \mathcal{EMB}(Id, m, k, h)$ // OAEP embed Id into curve
3: Look up most recent key $\mathbf{PD_{Pi}}$
4: Generate $t \in_R \mathbb{F}_q^*$ and compute $E = \mathcal{EG}_e(t, P_1, \mathbf{ID_{Pi}})$ // ElGamal encryption
5: $\mathtt{uDEI} = (\mathtt{uDEI_M}, E)$ // form unsigned DEI
6: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
7: Generate $\mathbf{SED_a}$ // see Appendix C.2
8: $AB_2 = \mathcal{E}_{\mathtt{AES}}(AB_1, \mathbf{SED_a})$ // encrypt AB for Supervisor
9: Represent uDEI, $AB_2, \tilde{T}$, $\mathbf{u}$.KV as byte array B // e.g. DER encoding
10: Compute $Sig = Sig_{\mathtt{dsa}}(B, \mathbf{u})$ // sign
11: Return DEI $= (\mathtt{DEI_M}, \mathtt{uDEI}, AB_2, \tilde{T}, Sig)$

In Step 4 of Algorithm 16 we simply let BSN-L choose the most recent key $\mathbf{PD_{Pi}}$. In practical implementations, BSN-L could have several registered several keys $\mathbf{PD_{Pi}}$ for a service provider, e.g. corresponding with various environments. To address this, Algorithm 16 could also take an identifier as extra input referring to the specific $\mathbf{PD_{Pi}}$ to be used.

We do not explicitly specify the validation and decryption of a DEI as this similar to that of EI. It consists of a input validation, a verification of the ECDSA signature followed with an ElGmamal decryption. Note that the occurrence of $\mathrm{AP_{ID}}$ and the timestamp T in a DEI have no technical reason, i.e. are not technically required in the actual validation and decryption process. The reason for their occurrence is allowing BSN-L verifyiing that the DEI provided by an authentication provider was indeed provided by BSN-L to that authentication provider and is sufficiently fresh.

5.2.5 Generation of Direct Encrypted Pseudonym (DEP)

A Direct Encrypted Pseudonym (DEP) is similar to that of an encrypted pseudonym (EP) but avoids the polymorphic transformation step which can be required in certain use cases. See Section 5.3.1. We let $\mathrm{DEP_M}, \mathrm{uDEP_M}$ denote the cryptogram metadata associated with a (unsigned) Direct Encrypted Pseudonym. Next to BSN-L two parties are involved in a DEP: the party requesting it and the intended service provider to which the DEP is finally sent and that can decrypt it. The intended service provider is indicated in Recipient field of

the metadata as introduced in Section 4.9. Also, the party authorized to use the DEP is reflected in an additional authorized party field in the DEP cryptogram metadata. In a DEP we allow for an optional role R in the direct (encrypted) pseudonym generation. This role takes the form of an optional field included in the DEP cryptogram metadata. The role will used as derivation data as indicated in Appendix C.2.

Usage of the DEP service is subject to an application process deciding if the service is allowed for a particular requester/service provider. If successful, the application process will lead to the issuance to BSN-L of a key of type $\mathbf{DT_{Di,R}}$ and the corresponding private key $\mathbf{DR_{Di,R}}$ to the intended service provider. See Appendix C.2.

The keys $\mathbf{DT_{Di,R}}$, $\mathbf{DR_{Di,R}}$ are closely related to the private key ($\mathbf{PD_{Di}}$) and public key ($\mathbf{PD_{Pi}}$) of a service provider related to encrypted pseudonyms, cf. 5.2.5. In practise it seems convenient to generate all four keys in concert for service providers that require both types of keys.

Algorithm 16 *GenDEP*($\mathrm{SP_{ID}}, Id$)
Generate Direct Encrypted Pseudonym for requesting party $\mathrm{SP_{IDr}}$ for intended service provider $\mathrm{SP_{IDi}}$ and role R based on Id.

1: $SN{=}SN{+}1$ // increment sequence number.
2: Compute $Q_1 = \mathcal{W}(\mathbf{IW_M}, Id)$ // keyed mapping of Id into curve
3: Compute $Q_2 = \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot Q_1$ // additional keyed mapping in curve
4: Look up most recent key $\mathbf{DT_{Di,R}}$
5: Compute $Q_3 = \mathbf{DT_{Di,R}}.\mathtt{Keyd[0]} \cdot Q_2$ // form half of pseudonym
6: Generate $t \in_R \mathbb{F}_q^*$ and compute $E = \mathcal{EG}_e(t, Q_3, \mathbf{DT_{Di,R}}.\mathtt{Keyd[1]})$ // base DEP
7: $\mathtt{uDEP} = (\mathtt{uDEP_M}, E)$ // form unsigned DEP
8: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
9: Generate $\mathbf{SED_a}$ // see Appendix C.2
10: $AB_2 = \mathcal{E}_{\mathrm{AES}}(AB_1, \mathbf{SED_a})$ // encrypt AB for Supervisor
11: Represent $\mathtt{uDEP}, AB_2, \tilde{T}, \mathbf{u}.\mathtt{KV}$ as byte array B // e.g. DER encoding
12: Compute $Sig = Sig_{\mathrm{dsa}}(B, \mathbf{u})$ // sign
13: Return $\mathtt{DEP} = (\mathtt{DEP_M}, \mathtt{uDEP}, E, AB_2, \tilde{T}, Sig)$

In Step 4 of Algorithm 16 we simply let BSN-L choose the most recent key $\mathbf{DT_{Di,R}}$. In practical implementations, BSN-L could have several registered several keys $\mathbf{DT_{Di,R}}$ for a service provider, e.g. corresponding with various environments. To address this, Algorithm 16 could also take an identifier as extra input referring to the specific $\mathbf{DT_{Di,R}}$ to be used.

5.3 Transformation of polymorphic to encrypted forms at APs

The role of authentication providers is to authenticate their users for service providers and resulting in an authentication message. After successful authentication an authentication provider transforms polymorphic forms (PI, PP or PIP) to encrypted forms (EI or EP). For this authentication providers maintain Hardware Security Modules (HSMs) to transform polymorphic forms to encryp-

ted forms. That is, polymorphic identities are transformed to encrypted identities and polymorphic pseudonyms are transformed to encrypted pseudonyms.

As indicated in Section 5.1, each HSM in the scheme is identified with a unique HSM_{ID} of 4 byte administered by the KMA. Moreover each HSM is able to produce a timestamp T (4 byte) and a sequence number SN (8 byte) starting with zero. For each encrypted form returned the HSM increments the sequence number. As indicated in Table 6 of Appendix C.1, an authentication provider is provided the following keys by the KMA:

- $\mathbf{AA_{Di}}$: an HMAC key called Authentication provider Adherence Derived key. All polymorphic forms (PI/PP/PIPs) are Authentication Specific and this key is required to use these forms.
- $\mathbf{IE_M}$: an HMAC key called Identity Encryption Master key it allows the transformation from Polymorphic Identity (PI) to Encrypted Identity (EI). The master key $\mathbf{IE_M}$ is used by the authentication provider during a transformation (authentication) to derive for each service provider an ephemeral key $\mathbf{IE_{Di}}$ called Identity Encryption Derived key. With this key, the authentication provider can rekey a polymorphic identity to a form decipherable by the service provider.
- $\mathbf{PE_M}$: an HMAC key called Pseudonym Encryption Master key. Together with the $\mathbf{PS_M}$ key it allows the transformation from Polymorphic Pseudonym (PP) to Encrypted Pseudonym (EP). The master key $\mathbf{PE_M}$ is used by the authentication provider during a transformation (authentication) to derive for each service provider an ephemeral key $\mathbf{PE_{Di}}$ called Pseudonym Encryption Derived key. With this key, the authentication provider can rekey a polymorphic pseudonym to a form decipherable by the service provider.
- $\mathbf{PS_M}$: an HMAC key called Pseudonym Shuffle Master key. Together with the $\mathbf{PE_M}$ key it allows the transformation from Polymorphic Pseudonym (PP) to Encrypted Pseudonym (EP). The master key $\mathbf{PS_M}$ is used by the authentication provider during a transformation (authentication) to derive for each service provider an ephemeral key $\mathbf{PS_{Di,R}}$ called Pseudonym Shuffle Derived key. With this key, the authentication provider can re-shuffle the base pseudonym in the polymorphic pseudonym for the service provider.

As indicated in Section 5.2 each polymorphic form is signed and contains an identifier (AP_{ID}) of the intended authentication provider. We assume that an HSM of an AP only accepts polymorphic forms intended for the AP. We consider this part of input validation that is explicitly specified in each HSM based algorithm.

Authentication provider processing rules
In the algorithms below, an authentication provider use keys most recent keys they possess with key indicators that compatible with the input cryptograms and keys. Compare the note after Algorithm 17. The actual keys used will be reflected in the KeyIndicator of the resulting cryptogram. See Section 4.9. As part of smooth key roll-over we also allow authentication providers to deviate from this in a controlled version. This will further elaborated on in Section 10.

5.3.1 Generation of Encrypted Identity (EI)

In this section we specify two algorithms transforming an PI or an PIP associated to a person into an encrypted identity for a service provider. The encrypted identity decrypts to global identifier of the person, typically its BSN. We let EI_{M} denote the cryptogram metadata associated with an Encrypted Identity. In Algorithms 17 and 18 we overload the notation *GenEI*() allowing it take both PI as PIP as input.

Algorithms 17, 18 take as input a polymorphic form PP or PIP and a service provider public key $\mathbf{ID_{Pi}}$. In Step 11 of these algorithms we require that the polymorphic forms and the public key are compatible. This effectively means that the private part of the public key $\mathbf{Y}$ used in PI, PIP, i.e. $\mathbf{y}$, is also used in the derivation of key $\mathbf{ID_{Pi}}$. Compare Appendix C.2. This compatibility plays an important role in key roll-over, cf. Section 10. We note that the key $\mathbf{SED_t}$ generated in Step 12 in Algorithms 17, 18 is independent of all other keys used in the generation of the encrypted identity. In line with the authentication provider processing rules (cf. end of Section 5.3), this means that the authentication providers uses the most recent one it has of the same version as the other keys.

Instead of the public key $\mathbf{ID_{Pi}}$ in Algorithms 17, 18 one could also as input the service provider identity $\mathbf{ID_{Pi}}$.Recipient and the $\mathbf{ID_{Pi}}$ key version (KVS) as these suffice to generate the Encrypted Identity; the public key $\mathbf{ID_{Pi}}$ then actually appears as the Encrypted Identity third point. Regardless of this; it is important that the input data is verified on authenticity, e.g. wrapped into or based on information in a digital certificate issued to the service provider.

Algorithm 17 *GenEI*($\mathrm{SP_{ID}}, \mathrm{PI}, \mathbf{ID_{Pi}}$)
$\mathrm{AP_{ID}}$ generates Encrypted Identity for $\mathrm{SP_{ID}}$ using polymorphic identity PI and service provider public key $\mathbf{ID_{Pi}}$.

1: SN=SN+1 // increment sequence number.
2: Validate that PI is correctly formed as $(\mathtt{PI_M}, \mathtt{uPI}, AB, \tilde{T}, Sig)$ and not expired, on failure return Error // input validation, assessment of $\tilde{T}$
3: Represent $\mathtt{uPI}, AB, \tilde{T}$ as byte array B // e.g. DER encoding
4: If $Ver_{\mathtt{dsa}}(B, Sig, \mathbf{U})$=False return Error // signature check
5: Interpret uPI as $(\mathtt{uPI_M}, A)$ with A an 3-tuple $(A_1, A_2, \mathbf{Y}) \in \mathbb{G}^3$ on failure return Error // parsing of PI
6: Compute $E_1 = \mathcal{RR}(A_1, A_2, \mathbf{Y})$ // randomize PI
7: Look up key $\mathbf{AA_{Di}}$ compatible with PI, on failure, return Error
8: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
9: Compute $\mathbf{IE_{Di}}$ based on $\mathbf{Y}, \mathbf{ID_{Pi}}$ // see Appendix C.2
10: Compute $E_3 = \mathcal{RK}(E_2, \mathbf{IE_{Di}})$ // rekey PI $\rightarrow$ Base EI
11: If third point in E_3 (public key) is not equal to $\mathbf{ID_{Pi}}$ return Error
12: $\mathtt{uEI} = (\mathtt{uEI_M}, E_3)$ // form unsigned EI
13: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
14: Generate $\mathbf{SED_t}$ // see Appendix C.2
15: $AB_2 = \mathcal{E}_{\mathtt{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
16: Represent $\mathtt{uEI}, AB_2, \tilde{T}$ as byte array B // e.g. DER encoding
17: Compute $Sig = Sig_{\mathtt{Schn}}(B, \mathbf{Y})$ // sign
18: Return $\mathtt{EI} = (\mathtt{EI_M}, \mathtt{uEI}, AB_2, \tilde{T}, Sig)$

Algorithm 18 *GenEI*($\mathrm{SP_{ID}}$, PIP, $\mathbf{ID_{Pi}}$)
$\mathrm{AP_{ID}}$ generates Encrypted Identity for $\mathrm{SP_{ID}}$ with polymorphic identity and pseudonym PIP and service provider public key $\mathbf{ID_{Pi}}$.

1: $SN{=}SN{+}1$ // increment sequence number.
2: Validate that PIP is correctly formed as $(\mathtt{PIP_M}, \mathtt{uPIP}, AB, \tilde{T}, Sig)$ and not expired, on failure return Error // input validation, assessment of $\tilde{T}$
3: Represent $\mathtt{uPIP}, AB, \tilde{T}$ as byte array B // e.g. DER encoding
4: If $Ver_{\mathtt{dsa}}(B, Sig, \mathbf{U}){=}\mathtt{False}$ return Error // signature check
5: Interpret uPIP as $(\mathtt{uPIP_M}, A)$ with A an 5-tuple $(A_1, A_2, A_3, \mathbf{Y}, A_5) \in \mathbb{G}^5$ on failure return Error // parsing of PIP
6: Validate that $\mathbf{Y}$ and $\mathbf{ID_{Pi}}$ are compatible, on failure return Error
7: Compute $E_1 = \mathcal{RR}(A_1, A_2, \mathbf{Y})$ // randomize PI
8: Look up key $\mathbf{AA_{Di}}$ compatible with PIP, on failure, return Error
9: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
10: Compute $\mathbf{IE_{Di}}$ based on $\mathbf{Y}, \mathbf{ID_{Pi}}$ // see Appendix C.2
11: Compute $E_3 = \mathcal{RK}(E_2, \mathbf{IE_{Di}})$ // rekey PI $\rightarrow$ Base EI
12: $\mathtt{uEI} = (\mathtt{uEI_M}, E_3)$ // form unsigned EI
13: If third point in E_3 (public key) is not equal to $\mathbf{ID_{Pi}}$ return Error
14: Form byte array $AB_1 = \mathtt{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
15: Generate $\mathbf{SED_t}$ // see Appendix C.2
16: $AB_2 = \mathcal{E}_{\mathtt{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
17: Represent $\mathtt{uEI}, AB_2, \tilde{T}$ as byte array B // e.g. DER encoding
18: Compute $Sig = Sig_{\mathtt{Schn}}(B, \mathbf{Y})$ // sign
19: Return $\mathtt{EI} = (\mathtt{EI_M}, \mathtt{uEI}, AB_2, \tilde{T}, Sig)$

5.3.2 Generation of Encrypted Pseudonym (EP)

In this section we describe two algorithms transforming an PP or an PIP associated to a person into an encrypted pseudonym for a service provider. From the encrypted pseudonym the service provider can deduce a specific pseudonym for the person. We let $\mathrm{EP_M}, \mathrm{uEP_M}$ denote the cryptogram metadata associated with an (unsigned) Encrypted Pseudonym. In Algorithms 19 and 20 we overload the notation *GenEP*() allowing it take both PP as PIP as input. These algorithms also allow incorporate an optional role R in the (encrypted) pseudonym generation. This role takes the form of byte array, which can be empty. We assume that this reflected in an additional field in the cryptogram data, i.e. the byte array *CrData* of Section 4.9.

Algorithms 19, 20 take as input a polymorphic form PP or PIP and a service provider public key $\mathbf{PD_{Pi}}$. In Step 3 of these algorithms we require that the polymorphic form and the public key $\mathbf{PD_{Pi}}$ are compatible. This effectively means that the private part of the public key $\mathbf{Z}$ used in PI, PIP, i.e. $\mathbf{z}$, is also used in the derivation of key $\mathbf{ID_{Pi}}$. Compare Appendix C.2. This compatibility plays an important role in key roll-over, cf. Section 10.

Instead of the public key $\mathbf{PD_{Pi}}$ in Algorithms 17, 18 one could also as input the service provider identity $PDPi$.Recipient and the key indicator $IDPi$.KI as these suffice to generate the Encrypted Identity; the public key $\mathbf{PD_{Pi}}$ then

actually appears as the Encrypted Identity third point. Regardless of this; it is important that the input data is verified on authenticity, e.g. wrapped into or based on information in a digital certificate issued to the service provider.

Algorithm 19 *GenEP*($\mathrm{SP_{ID}}, R, \mathrm{PP}, \mathbf{PD_{Pi}}$)
$\mathrm{AP_{ID}}$ generates Encrypted Pseudonym for $\mathrm{SP_{ID}}$ and optional role R with polymorphic pseudonym PP and service provider public key $\mathbf{PD_{Pi}}$.

1: SN=SN+1 // increment sequence number
2: Validate that PP is correctly formed as $(\mathrm{PP_M}, \mathrm{uPP}, AB, \tilde{T}, Sig)$ and not expired, on failure return Error // input validation, assessment of $\tilde{T}$
3: Represent uPP, $AB, \tilde{T}$ as byte array B // e.g. DER encoding
4: If $Ver_{\mathrm{dsa}}(B, Sig, \mathbf{U})$=False return Error // signature check
5: Interpret uPP as $(\mathrm{uPP_M}, A)$ with A an 3-tuple $(A_1, A_2, \mathbf{Z}) \in \mathbb{G}^3$ on failure return Error // parsing of PP
6: Validate that $\mathbf{Z}$ and $\mathbf{PD_{Pi}}$ are compatible, on failure return Error
7: Compute $E_1 = \mathcal{RR}(A_1, A_2, \mathbf{Z})$ // randomize PP
8: Look up key $\mathbf{AA_{Di}}$ compatible with PIP, on failure, return Error
9: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
10: Compute $\mathbf{PE_{Di}}$ based on $\mathbf{Z}, \mathbf{PD_{Pi}}$ // see Appendix C.2
11: Compute $E_3 = \mathcal{RK}(E_2, \mathbf{PE_{Di}})$ // rekey PP $\rightarrow$ Base EP.
12: If third point in E_3 (public key) is not equal to $\mathbf{PD_{Pi}}$ return Error
13: Compute $\mathbf{PS_{Di,R}}$ based on $\mathbf{PD_{Pi}}$ // see Appendix C.2
14: Compute $E_4 = \mathcal{RS}(E_3, \mathbf{PS_{Di,R}})$ // reshuffle PP $\rightarrow$ Base EP
15: $\mathrm{uEP} = (\mathrm{uEP_M}, E_4)$ // form unsigned EP
16: Form byte array $AB_1 = \mathrm{HSM_{ID}} || T || SN$ // form 16 byte Audit Block
17: Generate $\mathbf{SED_t}$ // see Appendix C.2
18: $AB_2 = \mathcal{E}_{\mathrm{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
19: Represent uEP, $AB_2, \tilde{T}$ as byte array B // e.g. DER encoding
20: Compute $Sig = Sig_{\mathrm{Schn}}(B, \mathbf{Z})$ // sign
21: Return $\mathrm{EP} = (\mathrm{EP_M}, \mathrm{uEP}, AB_2, \tilde{T}, Sig)$

Algorithm 20 *GenEP*($\mathrm{SP_{ID}}, R, \mathrm{PIP}, \mathbf{PD_{Pi}}$)
$\mathrm{AP_{ID}}$ generates Encrypted Pseudonym for $\mathrm{SP_{ID}}$ and optional role R with polymorphic identity and pseudonym PIP and service provider public key $\mathbf{PD_{Pi}}$.

1: $SN{=}SN{+}1$ // increment sequence number
2: Validate that PIP is correctly formed as $(\mathtt{PIP_M}, \mathtt{uPIP}, AB, \tilde{T}, Sig)$ and not expired, on failure return Error // input validation, assessment of $\tilde{T}$
3: Represent $\mathtt{uPIP}, AB, \tilde{T}$ as byte array B // e.g. DER encoding
4: If $Ver_{\mathrm{dsa}}(B, Sig, \mathbf{U}){=}$False return Error // signature check
5: Interpret uPIP as $(\mathtt{uPIP_M}, A)$ with A an 5-tuple $(A_1, A_2, A_3, A_4, \mathbf{Z}) \in \mathbb{G}^5$ on failure return Error // parsing of PIP
6: Validate that $\mathbf{Z}$ and $\mathbf{PD_{Pi}}$ are compatible, on failure return Error
7: Compute $E_1 = \mathcal{RR}(A_1, A_3, \mathbf{Z})$ // randomize PI
8: Look up key $\mathbf{AA_{Di}}$ compatible with PIP, on failure, return Error
9: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
10: Compute $\mathbf{PE_{Di}}$ based on $\mathbf{Z}, \mathbf{PD_{Pi}}$ // see Appendix C.2
11: Compute $E_3 = \mathcal{RK}(E_2, \mathbf{PD_{Pi}})$ // rekey PP $\rightarrow$ Base EP.
12: If third point in E_3 (public key) is not equal to $\mathbf{PD_{Pi}}$ return Error
13: Compute $\mathbf{PS_{Di,R}}$ based $\mathbf{PD_{Pi}}$ // see Appendix C.2
14: Compute $E_4 = \mathcal{RS}(E_3, \mathbf{PS_{Di,R}})$ // reshuffle PP $\rightarrow$ Base EP
15: $\mathtt{uEP} = (\mathtt{uEP_M}, E_4)$ // form unsigned EP
16: Form byte array $AB_1 = \mathtt{HSM_{ID}} \,||\, T \,||\, SN$ // form 16 byte Audit Block
17: Generate $\mathbf{SED_t}$ // see Appendix C.2
18: $AB_2 = \mathcal{E}_{\mathrm{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
19: Represent $\mathtt{uEP}, AB_2, \tilde{T}$ as byte array B // e.g. DER encoding
20: Compute $Sig = Sig_{\mathrm{Schn}}(B, \mathbf{Z})$ // sign
21: Return $\mathtt{EP} = (\mathtt{EP_M}, \mathtt{uEP}, AB_2, \tilde{T}, Sig)$

5.4 Validation and decryption at Service Providers

In Sections 5.4.1 - 5.4.2 we specify the algorithms allowing a Service Provider to validate and decrypt Encrypted Identities, Encrypted Pseudonyms and related cryptograms. If successful this results in the global identifier of the user, e.g. its BSN, or into a service provider specific pseudonym. In Section 5.4.5 we indicate how so-called PKCS #11 compliant devices can be used by service providers in decrypting. This facilitates the easy use of commonly used software libraries and (more importantly) Hardware Security Modules.

As indicated in Table 6 of of Appendix C.1, a service provider is provided the following keys by the KMA:

- $\mathbf{Y}$, $\mathbf{Z}$: These are ElGamal public keys called Identity Private Public key and Pseudonym Private Public key respectively. These are central public keys used in the creation of Polymorphic Identities and Polymorphic Pseudonyms by BSN-L but they also relevant for Service Providers for the verification of Schnorr signatures.
- $\mathbf{ID_{Di}}$: an EC private key (ElGamal decryption) called Identity Decryption Derived key.

- $\mathbf{ID_{Pi}}$: EC public key corresponding to $\mathbf{ID_{Di}}$ called Identity Decryption Public key
- $\mathbf{PD_{Di}}$: an EC private key (ElGamal decryption) called Pseudonym Decryption Derived key.
- $\mathbf{PD_{Pi}}$: EC public key corresponding to $\mathbf{PD_{Di}}$ called Pseudonym Decryption Public key.
- $\mathbf{PC_{Di}}$: an EC private key (Diffie-Hellman) called Pseudonym Closing Derived key.

A service provider that is only allowed entitled to user pseudonyms will not be provided $\mathbf{ID_{Di}}$ (and $\mathbf{ID_{Pi}}$) by the KMA. Moreover, a service provider, e.g. the user inspection service, entitled to Direct Encrypted Pseudonyms sent by BSN-L is provided an Direct Receiving Derived key, i.e. $\mathbf{DR_{Di,R}}$, by the KMA.

5.4.1 Validation and decryption of an Encrypted Identity

Algorithm 21 *DecEI*(EI)
Validation and decryption by service provider $\mathrm{SP_{ID}}$ of Encrypted Identity EI.

1: `Validate that EI is correctly formed as` $(\mathtt{EI_M}, \mathtt{uEI}, AB, \tilde{T}, Sig)$ `and not expired, on failure return Error // input validation, assessment of` $\tilde{T}$
2: `Interpret uEI as` $(\mathtt{uEI_M}, A)$ `with` A `an 3-tuple` $(A_1, A_2, \mathbf{ID_{Pi}}) \in \mathbb{G}^3$ `on failure return Error // parsing of EI`
3: `Represent uEI,` $AB, \tilde{T}$ `as byte array` B `// e.g. DER encoding`
4: `If` $Ver_{\mathrm{Schn}}(B, Sig, \mathbf{ID_{Pi}}, \mathbf{Y})$`=False return Error // signature check`
5: `Look up key` $\mathbf{ID_{Di}}$ `corresponding with` $\mathbf{ID_{Pi}}$
6: `Compute` $P = \mathcal{EG}_d(A_1, A_2, \mathbf{ID_{Pi}}, \mathbf{ID_{Di}})$ `// ElGamal decryption (Section 4.2)`
7: `Compute` $Id = \mathcal{DEC}(P, m, k, h)$ `// OAEP decoding`
8: `If last step was successful Return` Id `otherwise return Error`

We remark that by using Algorithms 5, 6 a service provider can demonstrate that (the ElGamal part of) an encrypted identity actually decrypts to a certain identity (BSN).

5.4.2 Validation and decryption of Direct Encrypted Identity

A DEI is generated by BSN-L for a requesting party allowing an intended service provider access to the user identity (BSN) similar as in an EI. A prominent use-case is where the requesting party is an authentication provider and the intended service provider is BSN-L itself, allowing the authentication provider to perform recurrent activation. Compare Section 5.2.4. The next algorithm specifies the validation and decryption of a DEI.

Algorithm 22 *DecDEI*(DEI)
Validation and decryption by intended service provider $\mathrm{SP}_{\mathrm{IDi}}$ of Direct Encrypted Identity DEI.

1: `Validate that DEI is correctly formed as` $(\mathtt{DEI_M}, \mathtt{uDEI}, AB, \tilde{T}, Sig)$ `and not expired, on failure return Error // input validation, assessment of` $\tilde{T}$
2: `Represent uDEI,` $AB, \tilde{T}$ `as byte array` B `// e.g. DER encoding`
3: `If` $Ver_{\mathtt{dsa}}(B, Sig, \mathbf{U})$`=False return Error // signature check`
4: `Interpret uDEI as` $(\mathtt{uDEI_M}, A)$ `with` A `an 3-tuple` $(A_1, A_2, \mathbf{ID_{Pi}}) \in \mathbb{G}^3$ `on failure return Error // parsing of DEI`
5: `Look up key` $\mathbf{ID_{Di}}$ `corresponding with` $\mathbf{ID_{Pi}}$
6: `Compute` $P = \mathcal{EG}_d(A_1, A_2, \mathbf{ID_{Pi}}, \mathbf{ID_{Di}})$ `// ElGamal decryption (Section 4.2)`
7: `Compute` $Id = \mathcal{DEC}(P, m, k, h)$ `// OAEP decoding`
8: `If last step was successful Return` Id `otherwise return Error`

5.4.3 Validation and decryption of an Encrypted Pseudonym

We let $\mathrm{P_M}$ denote the cryptogram metadata associated with a pseudonym optionally also indicating a role in a separate field.

Algorithm 23 *DecEP*(EP)
Validation and decryption by service provider $\mathrm{SP}_{\mathrm{ID}}$ of Encrypted Pseudonym EP for optional role R.

1: `Validate that EP is correctly formed as` $(\mathtt{EP_M}, \mathtt{uEP}, AB, \tilde{T}, Sig)$ `and not expired, on failure return Error // input validation, assessment of` $\tilde{T}$
2: `Interpret uEP as` $(\mathtt{uEP_M}, A)$ `with` A `an 3-tuple` $(A_1, A_2, \mathbf{PD_{Pi}}) \in \mathbb{G}^3$ `on failure return Error // parsing of EP`
3: `Represent uEP,` $AB, \tilde{T}$ `as byte array` B `// e.g. DER encoding`
4: `If` $Ver_{\mathtt{Schn}}(B, Sig, \mathbf{PD_{Pi}}, \mathbf{Z})$`=False return Error // signature check`
5: `Look up key` $\mathbf{PC_{Di}}$ `compatible with EP, on failure, return Error`
6: `Compute` $(B_1, B_2, \mathbf{PD_{Pi}}) = \mathcal{RS}(A_1, A_2, \mathbf{PD_{Pi}}, \mathbf{PC_{Di}})$ `// pseudonym closing`
7: `Look up key` $\mathbf{PD_{Di}}$ `corresponding with` $\mathbf{PD_{Pi}}$
8: `Compute` $P = \mathcal{EG}_d(B_1, B_2, \mathbf{PD_{Pi}}, \mathbf{PD_{Di}})$ `// ElGamal decryption (Section 4.2)`
9: `Return` $(\mathtt{P_M}, P)$

We note that Lines 5 and 7 follow the pattern of Algorithm 7 and a service provider can also use algorithms Algorithms 7 and 8 to demonstrate that (the ElGamal part of) an encrypted pseudonym actually decrypts to a certain pseudonym. We also note that one can reverse Lines 5 and 7 in Algorithm 5.4.3 by first decrypting $(A_1, A_2, \mathbf{PD_{Pi}})$ and then multiply the result with $\mathbf{PC_{Di}}$ which would save one multiplication. However, with the current order in Algorithm 23 one ensures that the contents of $(A_1, A_2, \mathbf{PD_{Pi}})$ never occurs in plaintext which is security beneficial.

5.4.4 Validation and decryption of Direct Encrypted Pseudonym

A DEP is generated by BSN-L for a service provider effectively taking the role of an EP. The next algorithm specifies the validation and decryption of a DEP at the service provider. The generation of the derived key $\mathbf{DR_{Di,R}}$ and its counterpart $\mathbf{DT_{Di,R}}$ are specified in Appendix C.2. We remark that both keys are generated in close relation with the decryption key $\mathbf{PD_{Di}}$.

Algorithm 24 *DecDEP*(DEP)
Validation and decryption by service provider $\mathrm{SP_{ID}}$ of Direct Encrypted Pseudonym DEP for role R.

1: `Validate that DEP is correctly formed as` $(\mathrm{DEP_M}, \mathrm{uDEP}, AB, \tilde{T}, Sig)$ `and not expired, on failure return Error // input validation, assessment of` $\tilde{T}$
2: `Represent uDEP,` $AB, \tilde{T}$ `as byte array` B `// e.g. DER encoding`
3: `If` $Ver_{\mathrm{dsa}}(B, Sig, \mathbf{U})$`=False return Error // signature check`
4: `Interpret uDEP as` $(\mathrm{uDEP_M}, A)$ `with` A `an 3-tuple` $(A_1, A_2, \mathbf{PD_{Pi}}) \in \mathbb{G}^3$ `on failure return Error // parsing of DEP`
5: `Compute` $(D_1, D_2, \mathbf{PD_{Pi}}) = \mathcal{RS}(A_1, A_2, \mathbf{PD_{Pi}}, \mathbf{DR_{Di,R}}.\mathrm{Keyd}[0])$
`// transform to regular EP`
6: `Look up key` $\mathbf{PC_{Di}}$ `compatible with DEP, on failure, return Error`
7: `Compute` $(E_1, E_2, \mathbf{PD_{Pi}}) = \mathcal{RS}(D_1, D_2, \mathbf{PD_{Pi}}, \mathbf{PC_{Di}})$ `// pseudonym closing`
8: `Look up key` $\mathbf{PD_{Di}}$ `corresponding with` $\mathbf{PD_{Pi}}$
9: `Compute` $P = \mathcal{EG}_d(E_1, E_2, \mathbf{PD_{Pi}}, \mathbf{PD_{Di}})$ `// ElGamal decryption (Section 4.2)`
10: `Return` $(\mathrm{P_M}, P)$

We note that one can combine Steps 4 and 5 with one re-shuffle operation with the key $\mathbf{DR_{Di,R}}.\mathrm{Keyd}[0] \cdot \mathbf{PC_{Di}}$, which saves two elliptic curve multiplications. That is, replacing Steps 4, 6 with one step:
Compute $(E_1, E_2, \mathbf{PD_{Pi}}) = \mathcal{RS}(D_1, D_2, \mathbf{PD_{Pi}}, \mathbf{DR_{Di,R}}.\mathrm{Keyd}[0] \cdot \mathbf{PC_{Di}})$.

For clarity reasons we have not done this. We also note that similar to the situation in Algorithm 23 one can allow a service provider to demonstrate that by using algorithms similar to Algorithms 7, 8 a service provider can demonstrate that (the ElGamal part of) a direct encrypted pseudonym actually decrypts to a certain pseudonym. This would consist of repeating Lines 2 of Algorithms 7 (respectively 8) with the Direct Receiving Derived key $\mathbf{DR_{Di,R}}$ and its corresponding public key $\mathbf{DR_{Di,R}} \cdot G$. The latter should be made available by the KMA in signed, i.e. verifiable, form.

5.4.5 PKCS #11 supported decryption at Service Providers

The PKCS #11 standard [42] specifies an application programming interface (API). This API specifies how an application can instruct a device holding cryptographic keys to perform certain cryptographic operations. These instructions only refer to the keys thus allowing the keys to be securely (non-exportably) stored in the device. PKCS #11 devices can be software (library) or hardware

based, e.g. smart cards or HSMs. The operations specified include commonly used cryptographic operations, although a PKCS #11 device is not required to implement them all. Modern PKCS #11 implementations include elliptic curve cryptography and Brainpool curves. This allows service providers to (securely) store their private decryptions keys ($\mathbf{ID_{Di}}, \mathbf{PD_{Di}}$) and closing keys ($\mathbf{PC_{Di}}$) in a PKCS #11 device. However, the ElGamal decryption and the closing operation are not part of the PKCS #11 instructions.

Clearly a PKCS #11 instruction to multiply any point P on the curve with a private key y would suffice for both operation. In Algorithm 25 below we indicate how the commonly supported `CKM_ECDH1_DERIVE` mechanism of PKCS #11 can be used to establish this. This mechanism is the core component of the Diffie-Hellman protocol and allows to instruct the device to calculate and return a hash value of the x-coordinate of $y{\cdot}P$. Moreover, PKCS #11 allows choosing the NULL hash function letting the device return the full x-coordinate of $y{\cdot}P$. We refer to this operation as `ECDH1_DERIVE_NULL`. Corresponding to this x-coordinate, two candidates for $y{\cdot}P$ exist and we are left with choosing the right candidate. This can be achieved by another call to `ECDH1_DERIVE_NULL` as indicated in Algorithm 25 below. Note that this algorithm not only uses the private key y but also the corresponding public key $Y = y{\cdot}G$.

Algorithm 25 $\mathcal{MP}_{11}(P, y, Y)$
PKCS #11 based multiplication of $P \in \mathbb{G}^*$ with private key y and public key Y.

1: `Check that` P, y, Y `are correctly formed` `// input validation`
2: `If` $P = G$ `return` Y `//` $P - G \neq \mathcal{O}$ `from now on`
3: `Compute` $r =$ `ECDH1_DERIVE_NULL`(P, y) `//` x`-coordinate of` $y{\cdot}P$
4: `Find point` $R \in \mathbb{G}$ `with x-coordinate equal to` r `//` $R = \pm Y$
5: `Compute` $s =$ `ECDH1_DERIVE_NULL`$(P - G, y)$ `//` x`-coordinate of` $y{\cdot}P - Y$
6: `Find point` $S \in \mathbb{G}$ `with x-coordinate equal to` s `//` $S = \pm(y{\cdot}P - Y)$
7: `If` $R - S = Y$ `Return` R
8: `If` $R - S = -Y$ `Return` $-R$
9: `If` $R + S = Y$ `Return` R
10: `If` $R + S = -Y$ `Return` $-R$

Proposition 5.1 *Algorithm 25 is correct, i.e. always returns* $y{\cdot}P$.

Proof: For $P = G$ Algorithm 25 is evidently correct. So without loss of generality we may assume that $P \neq G$. There are two possible outcomes of R computed at Step 4, namely $\pm y{\cdot}P$. Similarily, there are two possible outcomes of S computed at Step 6, namely $\pm(y{\cdot}P - Y)$. This means there are four possible outcomes of $R - S$ and $R + S$ as indicated in Table 1 below.

R	S	$R-S$	$R+S$
$y{\cdot}P$	$y{\cdot}P-Y$	Y	$2{\cdot}y{\cdot}P-Y$
$y{\cdot}P$	$-y{\cdot}P+Y$	$2{\cdot}y{\cdot}P-Y$	Y
$-y{\cdot}P$	$y{\cdot}P-Y$	$-2{\cdot}y{\cdot}P+Y$	$-Y$
$-y{\cdot}P$	$-y{\cdot}P+Y$	$-Y$	$-2{\cdot}y{\cdot}P+Y$

Table 1. Possible cases

Note that $2{\cdot}y{\cdot}P-Y=Y$ implies that $P=G$ which we assumed is not the case. Also note that $2{\cdot}y{\cdot}P-Y=-\mathrm{Y}$ implies that either $P=\mathcal{O}$ or $y=0$ both of which does not hold. Consequently, it follows that $2{\cdot}y{\cdot}P-Y\neq\pm Y$. This means that the four cases in lines 6-9 of Algorithm 25 precisely correspond and determine the four possible outcomes of R and S. Consequently this allows to determine the correct value for $y{\cdot}P$ from Table 1 as is done in lines 7-10 of Algorithm 25. □

5.5 Correctness proofs

In this section we prove correctness of the algorithms specified in Sections 5.2, 5.3 and 5.4.

Proposition 5.2 *The following hold in the context indicated in Figure 9:*

1. *The composition of Algorithms 12, 17 and 21 successfully returns the identity Id that was the input of Algorithm 12 to form the PI.*
2. *The composition of Algorithms 14, 18 and 21 successfully returns the identity Id that was the input of Algorithm 14 to form the PIP.*

Proof: We only show the first part of the proposition as the second part is analogous. For this we first focus on the ElGamal encryptions in the process in Figure 9, i.e. first disregarding the signatures generation/verifcations. To this end, in Step 2 of Algorithm 12 (PI generation) BSN-L forms the OAEP embedded identity $P_1\in\mathbb{G}$ and forms $P_2=\mathbf{AA_{Di}}^{-1}{\cdot}P_1$ in Step 4 of Algorithm 12. Compare Appendix C.2.

Next in Step 5 of Algorithm 12 the element P_2 is ElGamal encrypted under the scheme public key $\mathbf{Y}$ and placed as part of the PI outputted by BSN-L. This ElGamal encryption forms part of the input for the AP in Algorithm 17.

In Step 5 of Algorithm 17 this ElGamal encryption is randomized by the AP. By the first part of Proposition 4.1 this results in a random ElGamal encryption under public key $\mathbf{Y}$ of P_2. On this ElGamal encryption the AP next performs a reshuffling operation in Step 7 of Algorithm 17 using the key $\mathbf{AA_{Di}}$. According to the second part of Proposition 4.1 this results in an ElGamal encryption under $\mathbf{Y}$ of $\mathbf{AA_{Di}}\cdot P_2$. The latter is equal to $\mathbf{AA_{Di}}\cdot\mathbf{AA_{Di}}^{-1}\cdot P_1=P_1$. We conclude that the combination of Steps 5 and 7 results into a random ElGamal encryption under $\mathbf{Y}$ of P_1. In Step 9 of Algorithm 17 the AP next performs a

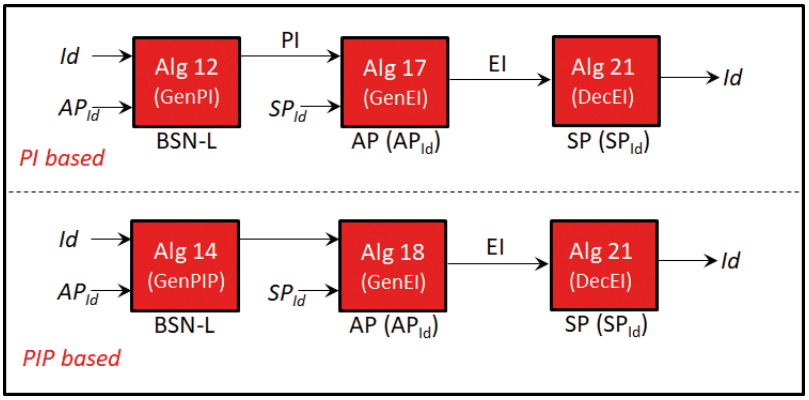

Figure 9. Polymorphic identity algorithm composition

re-keying operation with the key $\mathbf{IE_{Di}}$. By the third part of Proposition 4.1 this results in an ElGamal encryption of P_1 under the public key

$$\mathbf{IE_{Di}} \cdot \mathbf{Y}. \tag{18}$$

As $\mathbf{Y} = \mathbf{y} \cdot G$ (compare Table 6 of Appendix C.1) it follows that the public key in Formula (18) is equal to $\mathbf{ID_{Pi}}$, i.e. the EI public key of the SP. This ElGamal encryption is part of the EI output forming the input for the SP in Algorithm 21. In Step 5 of Algorithm 21, the SP uses his private key $\mathbf{ID_{Di}}$ corresponding to $\mathbf{ID_{Pi}}$ to obtain P_1. In Step 6 the SP performs an OAEP decoding leading to the identity *Id* that was input in Algorithm 12 of BSN-L forming the PI. We conclude that the ElGamal encryptions in the process behaves as implied in the proposition.

With respect to signatures in the process; in Step 3 of Algorithm 17 the AP performs an ECDSA signature verification on the received PI with public key $\mathbf{U}$. This verification is successful as this corresponds with the BSN-L signature generation on the PI with private key $\mathbf{u}$ in Step 7 of Algorithm 12. In Step 3 of Algorithm 21 the SP performs an EC-Schnorr signature verification on the received EI. This uses the EC-Schnorr public verification key $\mathbf{ID_{Pi}}$ with generator $\mathbf{Y}$. This corresponds to an EC-Schnorr signature with private key $\mathbf{IE_{Di}}$ and generator $\mathbf{Y}$. In Step 14 of Algorithm 17 such an EC-Schnorr signature is generated by the AP. □

Proposition 5.3 *The following hold in the context indicated in Figure 10:*

1. *The composition of Algorithms 13, 19 and 23 successfully returns a persistent pseudonym $P \in \mathbb{G}$ based on the identity Id that was the input of Algorithm*

12 to form the PI. The pseudonym $P \in \mathbb{G}$ takes the form

$$P = \underbrace{\mathbf{PC_{Di}}}_{SP} \cdot \underbrace{\mathbf{PS_{Di,R}}}_{AP} \cdot \underbrace{\mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \boldsymbol{\mathcal{W}}(\mathbf{IW_M}, Id)}_{BSN\text{-}L}. \tag{19}$$

Note: we have emphasized $\mathcal{W}$ in Formula (19) to indicate that its output is in $\mathbb{G}$; the factors are all in $\mathbb{F}_q^*$.

2. *The composition of Algorithms 14, 20 and 23 successfully returns a persistent pseudonym $P \in \mathbb{G}$ based on the identity Id that was the input of Algorithm 14 to form the PIP. The pseudonym $P \in \mathbb{G}$ takes the form as indicated in Formula (19).*

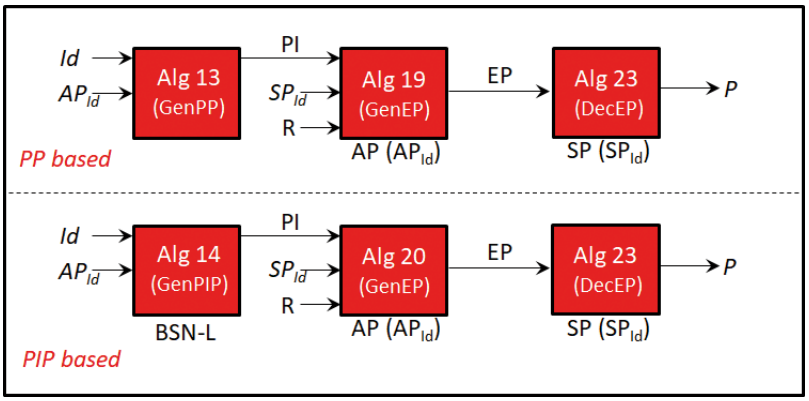

Figure 10. Polymorphic pseudonym algorithm composition

Proof: The proof resembles that of Proposition 5.2. We similarly focus on the first part as the second part is analogous. We also first disregard the signatures generation/verifcations, focussing on the ElGamal encryptions in the process in Figure 10. To this end, in Steps 2 and 3 of Algorithm 13 (PP generation) BSN-L forms the *base pseudonym* $Q_2 \in \mathbb{G}$. It is easily verified that

$$Q_2 = \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \mathcal{W}(\mathbf{IW_M}, Id). \tag{20}$$

In Step 5 the element $Q_3 = \mathbf{AA_{Di}}^{-1} \cdot Q_2$ is formed. Compare Appendix C.2. Next in Step 6 of Algorithm 13 the element Q_3 is ElGamal encrypted under the scheme public key $\mathbf{Z}$ and placed as part of the PP outputted by BSN-L. This ElGamal encryption forms part of the input for the AP in Algorithm 19.

In Step 6 of Algorithm 19 this ElGamal encryption is randomized by the AP. By the first part of Proposition 4.1 this results in a random ElGamal encryption under public key $\mathbf{Z}$ of Q_3. On this ElGamal encryption the AP next performs a

reshuffling operation in Step 8 of Algorithm 19 using the key $\mathbf{AA_{Di}}$. According to the second part of Proposition 4.1 this results in an ElGamal encryption under $\mathbf{Z}$ of $\mathbf{AA_{Di}} \cdot Q_3$. The latter is equal to $\mathbf{AA_{Di}} \cdot \mathbf{AA_{Di}}^{-1} \cdot Q_2 = Q_2$. We conclude that the combination of Steps 6-8 results into a random ElGamal encryption under $\mathbf{Z}$ of Q_2. In Step 10 of Algorithm 19 the AP next performs a re-keying operation with the key $\mathbf{PE_{Di}}$.

By the third part of Proposition 4.1 this results in an ElGamal encryption of Q_2 under the public key

$$\mathbf{PE_{Di}} \cdot \mathbf{Z}. \tag{21}$$

As $\mathbf{Z} = \mathbf{z} \cdot G$ (compare Table 6 of Appendix C.1) it follows that the public key in Formula (21) is equal to $\mathbf{PD_{Pi}}$, i.e. the EP public key of the SP. In Step 13 of Algorithm 19 the AP next performs a re-shuffling operation with the key $\mathbf{PS_{Di,R}}$. According to the second part of Proposition 4.1 this results in an ElGamal encryption under the service provider $\mathbf{PD_{Pi}}$ of

$$\mathbf{PS_{Di,R}} \cdot Q_2. \tag{22}$$

This ElGamal encryption is part of the EP output forming the input for the SP in Algorithm 23.

In Step 5 of Algorithm 23, the SP re-shuffles the received ElGamal encryption with his closing key $\mathbf{PC_{Di}}$. According to the second part of Proposition 4.1 this results in an ElGamal encryption under $\mathbf{PD_{Pi}}$ of

$$\mathbf{PC_{Di}} \cdot \mathbf{PS_{Di,R}} \cdot Q_2. \tag{23}$$

In Step 7 of Algorithm 23, the SP uses his private key $\mathbf{PD_{Di}}$ corresponding to $\mathbf{PD_{Pi}}$ to decrypt this ElGamal encryption. This leads to the element in $\mathbb{G}$ expressed in Formula (23). Combining this with Formula (20) it follows that the this element takes the form as indicated in Formula (19). This element pertains a persistent pseudonym as it only depends static keys, the identity of the user and that of the service provider.

With respect to signatures in the process; in Step 3 of Algorithm 19 the AP performs an ECDSA signature verification on the received PP with public key $\mathbf{U}$. This verification is successful as this corresponds with the BSN-L signature generation on the PI with private key $\mathbf{u}$ in Step 10 of Algorithm 13. In Step 3 of Algorithm 23 the SP performs an EC-Schnorr signature verification on the received EP. This uses the EC-Schnorr public verification key $\mathbf{PD_{Pi}}$ with generator $\mathbf{Z}$. This corresponds to an EC-Schnorr signature with private key $\mathbf{PE_{Di}}$ as in Formula (21) and generator $\mathbf{Z}$. In Step 17 of Algorithm 19 such an EC-Schnorr signature is generated by the AP. □

The proof of Proposition 5.3 until Formula (22) leads to the following corollary follows that will be useful later.

Corollary 5.1. *An Encrypted Pseudonym contains an ElGamal encryption under the public key* $\mathbf{PD_{Pi}}$ *of the service provider of the element*

$$\mathbf{PS_{Di,R}} \cdot \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \mathcal{W}(\mathbf{IW_M}, Id).$$

Proposition 5.4 *The composition of Algorithms 16 and 24 indicated in Figure 11 successfully returns the same persistent pseudonym P to a service provider as indicated in Formula (19). That is, Encrypted Pseudonyms and Direct Encrypted Pseudonyms lead to the same service provider pseudonyms.*

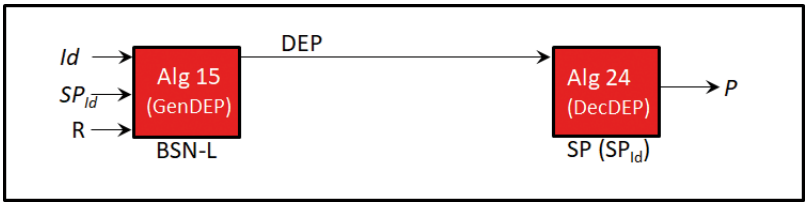

Figure 11. Direct pseudonym algorithm composition

Proof: The input and signature verifications by the service provider in Steps 1-3 of Algorithm 24 should be successful as they correspond with the DEP forming and signature generation by BSN-L in Algorithm 16. For the encryption part of the result we show that Step 4 in Algorithm 24 transforms an ElGamal encryption inside the DEP to an ElGamal encryption inside an EP for the service provider. As Steps 5-9 in Algorithm 24 coincide with Steps 4-8 in Algorithm 23 the result then follows.

To this end, it follows from steps 5 and 6 of Algorithm 16 that the ElGamal encryption in a DEP is that of the element

$$\mathbf{DT_{Di,R}}.\text{Keyd}[0] \cdot \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \mathcal{W}(\mathbf{IW_M}, Id) \tag{24}$$

under the public key $\mathbf{DT_{Di,R}}.\text{Keyd}[1]$. This public key is of the form $\mathbf{PD_{Pi}}$ of the service provider by Appendix C.2. This ElGamal encryption is part of the DEP output forming the input for the service provider in Algorithm 24. In Step 4 of Algorithm 24 a re-shuffling is performed by the service provider using the key $\mathbf{DR_{Di,R}}.\text{Keyd}[0]$. According to the third part of Proposition 4.1 this results into an ElGamal encryption under the public key $\mathbf{PD_{Pi}}$ of the service provider of the multiplication of the original contents, i.e. Formula (24), and $\mathbf{DR_{Di,R}}.\text{Keyd}[0]$. That is, Step 4 of Algorithm 24 results into an ElGamal encryption under the public key $\mathbf{PD_{Pi}}$ of the service provider of the element

$$\mathbf{DR_{Di,R}}.\text{Keyd}[0] \cdot \mathbf{DT_{Di,R}}.\text{Keyd}[0] \cdot \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \mathcal{W}(\mathbf{IW_M}, Id),$$

which is equal to

$$\mathbf{PS_{Di,R}} \cdot \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \mathcal{W}(\mathbf{IW_M}, Id)$$

by the construction of $\mathbf{DR_{Di,R}}.\text{Keyd}[0], \mathbf{DT_{Di,R}}.\text{Keyd}[0]$ in Appendix C.2. It now follows from Corollary 5.1 that Step 4 in 24 produces an ElGamal encryption equal to the one inside an EP. □

We define the *unclosed* pseudonym at a service provider as the plaintext payload inside an encrypted pseudonym, i.e. the point indicated in Formula (19) except the factor $\mathbf{PC_{Di}}$. The essence of DEPs is that they allow BSN-L to directly deliver pseudonyms to service providers without getting access to the unclosed pseudonyms and thus also to the final pseudonyms themselves. For future reference we formulate this as a corollary to the proof of Proposition 5.4.

Corollary 5.2. *The unclosed pseudonym resulting from a DEP is formed as a two party computation by BSN-L and the intended service provider. BSN-L does not get access to this unclosed pseudonyms resulting from the DEP.*

6 Formalization of the polymorphic eID scheme

In this section we now formalize the polymorphic eID scheme introduced in Section 2.2 using the cryptographic building blocks from Section 5. For this we formalize the protocols Authenticator Activation, Authentication (Transformation) and Authenticator Deactivation from Section 2.2. In these descriptions we also deploy a Transaction Logging Provider (TLP). Here the authentications transactions are recorded under pseudonym and available for the user. The TLP could be a separated part of the AP or could be placed at a different party. In all these protocols we assume that all (web) communication is suitably protected, e.g. through (double sided) TLS.

It easily follows from Propositions 5.2 and 5.3 that these protocols (most notably Authentication) meet the required functionality. Indeed, it follows from the first part of Proposition 5.2 that the *Id* (BSN) that was presented during activation by the AP is delivered to a service provider during identity based authentication. From Proposition 5.3 it follows that a persistent, service provider specific pseudonym is delivered to a service provider during pseudonymous authentication. The pseudonym takes the form as depicted in Formula (19) which is independent of the AP that helped to generate it. The protocols specified also deploy Direct Encrypted Pseudonyms (DEPs). That this approach results into the same pseudonyms as by using EPs is Proposition 5.4.

6.1 Polymorphic Authenticator Activation

In Protocol 1 we formalize polymorphic activation of a user AP authenticator.

Protocol 1 Authenticator issuance and activation
User is issued authenticator at AP which is activated in scheme.

1: AP verifies that user has identity *Id* in line with [19]
2: If Line 1 fails, then the protocol fails (and ends)
3: AP sends *Id* and other data *D* to BSN-L and requests PI+PP or PIP
4: BSN-L validates *Id* in combination with *D*
5: If Line 4 fails, then the protocol fails (and ends)
6: BSN-L forms AP specific PI+PP or PIP by Algorithms 12, 13, 14

```
 7: BSN-L sends PI+PP or PIP to AP in response to request in Line 3
 8: AP records PI+PP or PIP in registry under internal user-id
    // BSN-L transaction writing at UIS
 9: BSN-L forms DEP@UIS based on Id for UIS by Algorithm 16
10: BSN-L sends DEP@UIS and AP activation message to UIS
11: UIS determines the the user pseudonym from DEP@UIS by Algorithm 24
12: UIS records AP activation under user pseudonym
    // AP issuing authenticator
13: AP records new authenticator in registry under internal user-id
14: AP delivers authenticator to user in line with [19]
15: AP deletes Id and other data no longer required
    // AP transaction writing at UIS
16: AP selects PP of user and forms EP@UIS by Algorithm 19
17: AP sends EP@UIS and authenticator activation message to UIS
18: UIS determines the user pseudonym from DEP@UIS by Algorithm 23
19: UIS records authenticator activation under user pseudonym
    // AP transaction writing at TLP
20: AP selects PP of user and forms EP@TLP by Algorithm 19
21: AP sends EP@TLP and authenticator activation message to TLP
22: TLP determines the user pseudonym from EP@TLP by Algorithm 23
23: TLP records authenticator activation under user pseudonym
```

We make some remarks on Protocol 1. In Line 3 we let the AP send the user *Id*, i.e. BSN, to the BSN-L. This can be avoided by letting the AP send "enough" identifying data (e.g., first and last name, date and place of birth) to BSN-L to have a unique BSN match in the population register. This approach has as drawback that it introduces a non-uniqueness match risk and thus failure of Protocol 1. In Protocol 1 actually two things happen. In Lines 1-12 the authentication provider is activated in the scheme by BSN-L and in Lines 13-23 the AP authenticator is activated in the scheme. If the authentication provider would provide multiple authenticators to a user, Lines 1-12 would not have to be repeated. To support this one could split Protocol 1 into an "AP activation" and an "authenticator activation" part. For simplicity reasons we have not done so. The authenticator information in Lines 17 and 21 should enalbe users to distinguish authenticators when the user is issued several by the AP, e.g. an authentication APP and a hardware token. In practice this means that the authenticator information includes some serial number digits visible on/in the authenticator. Protocol 1 separates user and usage data in line with Section 2.2. Indeed, user data is recorded in a user registry, cf. Line 8, and usage is data is stored in the TLP. Of course it should be practically possible for an authenticator provider to setup a telephone user help desk. In case the authenticator fails, a help desk employee should be able to authenticate a user e.g. by asking questions. So the user registry should support that. More fundamentally, with consent of the user the help desk employee should be able to view authentication transactions in the TLP. For this the AP can deploy a TLP query service for the help desk which generates a user EP@TLP on the fly. This query service may only be used with user consent. To detect abuse the service should also write query transactions in the TLP logs that indicate queries (help desk employee, date/time, transac-

tion period) conducted. This allows the user to detect unauthorized queries by reviewing the TLP. We finally note that in Protocol 1 we could also distinguish APs only allowed to receive PPs. For this, BSN-L would only provide a PP in Line 7 of Protocol 1.

6.2 Polymorphic Authentication (Transformation)

In Protocols 2 and 3 we formalize polymorphic authentication to a service provider through an authenticator activated by an authentication provider through Protocol 1.

Protocol 2 *Authentication providing identity*
User authenticates to Service Provider whereby providing his identity through Authentication Provider.

```
 1: User wants to authenticate to SP providing his identity
 2: SP directs user to AP with request for identity based authentication
 3: User authenticates to AP using authenticator activated in Protocol 1
 4: User consents to AP on providing his identity to SP.
 5: On failure of Lines 3 or 4 AP redirects user to SP with failure response
 6: AP selects PI of user and forms EI@SP for SP by Algorithm 17
 7: User is redirected to SP with EI@SP in response
 8: SP determines the user identity from EI@SP by Algorithm 21
 9: On success, user is authenticated under his identity
10: On failure, user authentication at SP fails
    // AP transaction writing at TLP
11: AP selects PP of user and forms EP@TLP by Algorithm 19
12: AP sends EP@TLP and authentication transaction to TLP
13: TLP determines the user pseudonym from EP@TLP by Algorithm 23
14: TLP records authentication transaction under user pseudonym
```

Protocol 3 *Authentication providing pseudonym*
User authenticates to Service Provider whereby providing (role based) pseudonym through Authentication Provider.

```
 1: User wants to authenticate to SP providing (role based) pseudonym
 2: SP directs user to AP with request for pseudonym based authentication
 3: User authenticates to AP using authenticator activated in Protocol 1
 4: Optionally, the user chooses role in which to authenticate to SP
 5: User consents to AP on providing (role based) pseudonym to SP.
 6: On failure of Lines 2 or 4 AP redirects user to SP with failure response
 7: AP selects PP of user and forms EP@SP by Algorithm 19
 8: User is redirected to SP with EP@SP in response
 9: SP determines the user pseudonym from EP@SP by Algorithm 23
10: On success, user is authenticated under his pseudonym at SP
11: On failure, user authentication protocol at SP fails
    // AP transaction writing at TLP
12: AP selects PP of user and forms EP@TLP by Algorithm 19
13: AP sends EP@TLP and authentication transaction to TLP
14: TLP determines the user pseudonym from EP@TLP by Algorithm 23
```

15: TLP records authentication transaction under user pseudonym

Support for end-to-end security and privacy
At the end of Section 1.1 we introduced the routing services envisioned in the Dutch eID scheme. We also expressed the importance of providing end-to-end security and privacy between the authentication provider and the service provider. Polymorphic authentication specified above conveniently supports both. Indeed, end-to-end privacy is provided by letting the authentication protocols of the service providers and authentication providers only be used as a transport (envelop) mechanism for encrypted identities and encrypted pseudonyms. In this fashion, the routing service only retrieves an encrypted identity (or pseudonym) from the authentication response of the authentication provider and then integrally places this in a new authentication response for the service provider. By the properties of encrypted identities and encrypted pseudonyms the routing service does not get access to the identity or pseudonym inside. Actually, even the authentication provider does not get access to these. To support end-to-end security we need the service provider to include a random nonce in its authentication request to the routing service. This nonce is then also included in the authentication request of the routing service to the authentication provider. The latter then includes this in the scope of the Schnorr signature of the encrypted identity (Step 6 of Protocol 2 or encrypted pseudonym (Step 7 of Protocol 3). In this way the service provider can check that the nonce is part of the (signed) encrypted identity or encrypted pseudonym allowing him to conclude these are fresh. This usage of a nonce is known as a *challenge-response protocol* which is commonly used in authentication protocols. Note that the end-to-end security and privacy properties remain intact if more than only one proxy is deployed.

6.3 Polymorphic Authenticator Deactivation

In Protocol 4 we formalize deactivation of a user AP authenticator.

Protocol 4 *User Authenticator Deactivation*
User deactivates authenticator in scheme.

1: User contacts AP and requests for authenticator deactivation
2: AP selects PP of user and forms EP@UIS by Algorithm 19.
3: AP sends EP@UIS and authenticator deactivation message to UIS
4: UIS determines the user pseudonym from EP@UIS by Algorithm 23
5: UIS records authenticator deactivation at AP under user pseudonym
6: AP selects PP of user and forms EP@TLP by Algorithm 19
7: AP sends EP@TLP and authenticator deactivation transaction to TLP
8: TLP determines the user pseudonym from EP@TLP by Algorithm 23
9: TLP records deactivation transaction under user pseudonym
10: AP records authenticator deactivation in registry under internal user-id

The first line of Protocol 4 could be conducted by letting the user authenticate to the AP using the authenticator issues by the AP. However, other mechanisms

could also be in place such as a face-to-face contact. In the notes proceeding Protocol 1 we observed that activation consists of an AP and an authenticator part. In Protocol 4 we only deactivate the authenticator but we can similarly introduce AP deactivation. As part of AP deactivation one could also require the AP removing the polymorphic forms acquired during activation. This would then also lead to UIS transactions allowing the user to verify AP deactivation. Moreover, UIS would then no longer accept new AP authenticator activations. Such AP deactivation would be appropriate when the AP subscription of the user ends.

7 The polymorphic card application (PCA)

7.1 Introduction

The idea of the polymorphic card application (PCA) is that the AP does not store a PIP in a database as in Section 6 but places it in a card application. This PIP (or only its PP part) card application is then read by the AP from the card whereby the card first randomizes it. This randomization allows an AP to perform authentications in a practically anonymous way while still being able to deliver the BSN (or pseudonym) to service providers. As randomization makes PIP signatures delivered by BSN-L invalid, PCA works with unsigned PIPs (uPIPs) and PPs (uPPs). These forms are similar to the signed versions but lack the audit block and signature. The AP can assess PIP/PP authenticity and integrity from the secure channel ("EAC") over which it is read. This is similar to the setup within the German eID card (nPA), cf. Appendix F. The transformation algorithms performed by the PCA HSM only differ from these in Section 5.3 in that that they do not verify the PIP signature. For completeness we have specified the PIP transformation algorithms in Appendix D.

PCA is heavily based on the *neue Personalausweis (nPA)* card application residing on the German eID card. In a cooperation between the German and French government, an nPA extension has been developed called the eIDAS token, cf. [5,6,7]. A PCA instance is a two-factor authenticator consisting of a physical factor ("PCA card") and a knowledge factor ("PCA PIN"). An alternative name for PCA could be the Polymorphic eIDAS Token. We base the PCA description on the terminology and working of nPA. For convenience of the reader we have outlined these in Appendix F. Here we also discuss the eID client software installed by the user and the eID server software that needs to be installed by the authentication provider. In PCA context the authentication provider role is split into three different roles:

PCA card issuer (CI) This party registers the user, verifies its identity and requests polymorphic forms (PIP) from BSN-L like a regular authentication provider in Section 6. However, instead of storing it in a database the PIP is placed by the issuer on a PCA card protected under PIN. The PCA card and PIN are then securely delivered to the user in conformity with [18,19]. Unlike a regular authentication provider the issuer is not provided

the cryptographic keys by KMA required for the transformation function. Instead, these keys are provided to PCA authentication providers that are allowed to use the card (see below). Such authentication providers are also provided so-called terminal certificates by the issuer, cf. Appendix F. These allow an authentication provider to read PIPs (or only the PP part) from the PCA instance that first randomizes them. This operation requires that the user consents by entering his PCA PIN. Typically the issuer also provides a revocation password to the user allowing to revoke the card at the issuer; other mechanism could also be deployed. Compare [19].

PCA status service (SS) This is a pseudonymous service provider that maintains the status of cards produced by the card issuer. By usage of the service an authentication provider is ensured that a PCA instance is 1) produced and 2) not revoked. For similar functionality the German eID card deploys service provider specific "white" and "black" list respectively. The "white" lists are available but will only be distributed when serious card security flaws occur. See Appendix F. PCA standard supports both "white" and "black" listing. Supported statuses minimally include: "Produced", "Activated for AP AP_{ID}" and "Revoked". Various other card statuses could also be recorded in the status servie, e.g. "In Transit" (to user) and "Suspended".

PCA authentication providers These parties are allowed by the card issuer to authenticate users for service providers using the PCA cards. For this these parties are provided suitable terminal certificates provided by the card issuer and key material from the KMA. Authentication consists of reading a full PIP (respectively only the PP part) from a PCA card and transforming them into EI (respectively EP) as formalized in Section 6.2. There can be several authentication providers that can read the PCA card of a user.

The status service role is independent and segregated ('Chinese wall') from the other two roles. We consider the issuer to be the responsible party for the cards and the personal data that resides on it. In line with this we only allow the issuer to write to the PCA status service and allow authentication provider only to read from this service. Consequently, authentication providers need to interact with the card issuer for status service updates. Compare Protocol 7. In Sections 7.2 7.3 and 7.4 we specify the protocols Activation, Authentication (Transformation) and Deactivation for a PCA instance (authenticator). We already discussed these protocols in Section 6 in the generic polymorphic eID setup. In addition we specify PCA revocation in Section 7.5. In all these protocols we assume that all (web) communication is suitably protected, e.g. through (double sided) TLS. That PCA delivers the same polymorphic functionality as a regular PCA authenticator similarly follows from Propositions 5.2, 5.3 and 5.4 as indicated in Section 6.

7.2 PCA Activation

As in the German eID card (nPA, see Appendix F) batches of PCA cards share a CA key signed by an issuer document signer certificate that is also shared.

This setup is indicated in Protocol 5. Two configuration parameters, Batch_Size, Usage_Limit and running parameters b, μ play an important role in this protocol. The configuration parameter Batch_Size indicates the number of PCA cards in a batch, e.g. 100.000. The running parameter $1 \leq b \leq$ Batch_Size indicates the batch sequence number of a card within the issuer production process. As the status service is based on pseudonyms, all PCA cards of a user will relate to the same pseudonym. So by only using this pseudonym the status service is not able to distinghuish several PCA cards of a user at the same issuer. To do so we need to accompany every PCA instance of a user at an issuer with some limited extra data. This is addressed by the configuration parameter Usage_Limit. This indicates the number of cards one user may be issued within one batch. This number is considered small, e.g. 5. The extra information sent to the status service is indicated with the running parameter μ generated in Protocol 5 by Algorithm 26. As can be seen from Algorithm 26, μ is based on the issuance date of the document signer certificate and an integer not exceeding Usage_Limit. As this certificate is already revealed to the authentication provider during PCA authentication, this date does not add to extra distinguishably. From the discussion above, it is vital that the card issuer can recognize that the same user, i.e. same identity (BSN), is requesting several PCA cards within one batch. The simplest way to achieve this, is to let the card issuer register the user identity in its internal database. However, this was one of the things that was to be avoided. To this end, we let issuer database be based on pseudonyms too and let the issuer be provided a DEP by BSN-L as part of activation. We therefore assume that the card issuer is provided the secret (D)EP decryption and closing keys by the KMA, i.e. keys of type $\mathbf{DR_{Di,R}}, \mathbf{PD_{Di}}, \mathbf{PC_{Di}}$ and the BSN-L signature verification key **U** (cf. Table 6 of Appendix C.1).

We already remarked that in the context of PCA, the authentication provider role is split over the issuer and the PCA authentication provider. This also applies to Activation; before a PCA card can be used it needs to be activated by both the issuer and a PCA authentication provider. This is described in Protocols 5 and 7 below respectively. Within the first protocol the production of the card plays an important role; this is separately specified in Protocol 6.

Protocol 5 *PCA Activation at issuer*
PCA card issuance and activation by Card Issuer $\mathrm{CI_{ID}}$ for user with identity *Id*.

1: CI verifies that user has identity *Id* in line with [19]
2: CI sends *Id* and other data *D* to BSN-L and requests PIP, DEP@[CI,SS,UIS]
3: BSN-L validates *Id* in combination with *D*
4: If Line 3 fails, then the protocol fails (and ends)
5: BSN-L generates CI specific PIP by Algorithm 14
6: BSN-L generates DEP@[CI,SS,UIS] by Algorithm 16
7: BSN-L sends PIP, DEP@[CI,SS,UIS] to CI in response to request in Line 2
// BSN-L updates UIS
8: BSN-L generates new DEP@UIS based on *Id* for UIS by Algorithm 16
9: BSN-L informs UIS on CI activation referring to DEP@UIS
10: UIS determines the pseudonym of user from DEP@UIS by Algorithm 24

11: UIS records CI activation at UIS user pseudonym
// CI card production
12: CI determines pseudonym P of user from DEP@CI by Algorithm 24
13: CI tries producing PCA card, PIN, uniqueness identifier μ by Protocol 6
14: If Line 13 fails, then the protocol fails (and ends) // exception
15: CI generates revocation password R
16: CI registers card, μ, DEP@[SS,UIS] and R in database under pseudonym P
// CI updates SS
17: CI registers card ''Produced'' and μ at SS referring to DEP@SS
18: SS determines the user pseudonym P' from DEP@SS by Algorithm 24
19: SS registers card with uniqueness identifier ''Produced'' using P', μ
// CI updates UIS
20: CI sends DEP@UIS and CI PCA activation message to UIS
21: UIS determines the user pseudonym from DEP@UIS by Algorithm 24
22: UIS records CI activation transaction under user pseudonym
23: CI deletes Id and other data no longer required
24: CI delivers card, PIN and R to user in line with [19]

The information in Line 18 should allow users to distinguish PCA instances when the user is issued several, e.g. an old and a new one. In practice this means that this includes some document number digits visible on the PCA card. Note that the uniqueness identifier μ only needs to "inside" the PCA card and needs not be printed on it.

Protocol 6 *PCA card production by issuer*
PCA card production by Card Issuer with PIP = $(\text{PIP}_{\text{ID}}, \text{CI}_{\text{ID}}, A, B, C)$ and uniqueness identifier μ.

1: $k = k + 1$ // increase issued PCA cards in batch with one
2: **if** $k >$ Batch_Size **then** // get new batch
3: Let $k = 1$ // card will be first in new batch
4: Generate new DS key pair $(D_{\text{priv}}, D_{\text{pub}})$
5: Bind D_{pub} in DS certificate C_{DS}
6: Generate new shared CA public/private key pair $(P_{\text{priv}}, P_{\text{pub}})$
7: Bind P_{pub} in shared CA certificate C_{CA}
8: **end if**
9: Generate uniqueness identifier $\mu = U(P, C_{\text{DS},k})$ by Algorithm 26
10: If Line 9 is unsuccessful the protocol fails (and ends) // exception
11: Initiate and open new card in personalization machine
12: Place PCA instance on card
13: Place TA trust anchor in PCA instance. // CVCA
14: Generate random PIN and place this as PACE secret in PCA instance
15: Place DS and CA certificates $C_{\text{DS}}, C_{\text{CA}}$ in PCA instance
16: Place CA private key P_{priv} in PCA instance, in non-exportable form
17: Validate that $(\text{PIP}_{\text{ID}}, \text{CI}_{\text{ID}}, A, B, C)$ is correctly formed,
on failure return Error // input validation
18: If $Ver_{\text{dsa}}(\text{PIP}_{\text{ID}} || \text{AP}_{\text{ID}} || A || B, C, \mathbf{U}) =$ False return Error // signature check
19: Form uPIP $= (\text{uPIP}_{\text{ID}}, \text{CI}_{\text{ID}}, A)$ // strip PIP
20: Place uPIP and μ in PCA instance
21: Close card

22: `Return card, PIN and` μ

The information an authentication provider can read from PCA instances produced in the same batch coincides. The uPIP and μ in Line 20 are placed in secure memory and are not directly readable by a PCA authentication provider. These are only readable as part of a PCA authentication whereby the PIP (or only its PP part) is randomized. We remark that the relevant standards, e.g. [26], [5], do not actually wrap the CA key into a regular certificate. However, effectively one can consider this to be the case.

As part of quality control it is customary that a card issuer tests a produced card directly after production, i.e. following Step 22 of Protocol 6. In PCA context this ideally would be a complete simulation of the authentication process with the freshly produced card. However due to various security, privacy and operational risks this is not considered acceptable. Appendix E elaborates on this topic and also specifies the technique of Verifiable Polymorphic Identity and Pseudonym (VPIP) which provides the quality control functionality of the simulation without its risks.

Algorithm 26 $U(P, C_{\mathrm{DS}}, k)$
Generate PCA card uniqueness identifier for user with pseudonym P based on document signer certificate C_{DS} from issuer $\mathrm{CI}_{\mathrm{ID}}$.

1: `Let string` D `represent the issuance date of the DS certificate in form ''YYYYMMDD''`
2: `Let` S `be the set of` $i \in \{1, \ldots,$`Usage_Limit`$\}$ `such that entry` $D||i$ `is not registered in the issuer database under` P
3: `If` $\#(S) = 0$ `then return error` `// too many cards issued to user`
4: `Else if` $\#(S) =$ `Usage_Limit then` $i = k \bmod$ `Usage_Limit`
5: `Else` $i \in_R S$
6: `Register entry` $D||i$ `in the issuer database under` P
7: `Return` $\mu = \mathtt{CI_{ID}} \,|||\, D \,|||\, i$

Prior to a PCA authentication provider being able to allow users to activate and user their PCA card the following two conditions need to be satisfied. First, the authentication provider needs to be provided suitable TA certificates by the issuer allowing to read randomized PIPs or PPs from the card. Second, the authentication provider needs to be provider cryptographic keys by the KMA required to transform these to EIs, EPs or both. For the latter, the provider needs to be provided the public BSN-L signature verification key **U** and secret keys $\mathbf{AA_{Di}}$ (Authentication provide Adherence Derived key), $\mathbf{IE_M}$ (Identity Encryption Master key), $\mathbf{PE_M}$ (Pseudonym Encryption Master key and $\mathbf{PS_M}$ (Pseudonym Shuffle Master key). The key $\mathbf{AA_{Di}}$ is derived from the identifier of the card issuer, i.e. from $\mathrm{CI}_{\mathrm{ID}}$. As in Section 6 we let the PCA authentication providers deploy a Transaction Logging Provider (TLP). We only use one TLP but every PCA authentication provider could have its own TLP. This PCA authentication provider activation protocol 7 below is very similar to PCA authentication. As mentioned before authentication providers are not allowed to

write to the status service. In the Protocol 7 the authentication provider therefore requests the card issuer to update the status service. This request is based on the user pseudonym in the card issuer domain which is already used by the issuer in Protocol 5. The Protocol 7 refers to an eID client and eID server which are existing building within the German eID card setup and which are also outlined in Appendix F.

Protocol 7 *PCA card activation at AP*
PCA card activation at PCA authentication provider

1: User connects to the activation service of PCA authentication provider
2: User enters PIN, allows eID client to establish local PACE SM with PCA
3: Authentication provider sends TA certificate to eID client
4: eID client arranges EAC SM between PCA and AP eID server
5: User consents to AP on activation
6: eID client enforces that AP eID server only reads randomized unsigned uPP and uniqueness identifier μ from PCA
7: AP uses uPP to form EP@CI by Algorithm 37 (Appendix D)
8: AP sends PCA activation message to CI referring to EP@CI and μ
9: CI determines the user pseudonym P_1 from EP by Algorithm 23
10: CI uses P_1, μ, to check in database if PCA instance can be activated
11: If Line 9 fails then the protocol fails (and ends)
12: CI records AP activation under P_1, μ in database
// CI updates SS
13: CI selects DEP@SS from database
14: CI sends PCA activation message to SS referring to DEP@SS and μ
15: SS determines the user pseudonym P_2 from DEP@SS by Algorithm 24
16: SS records AP activation transaction and μ under P_2
// CI updates UIS
17: CI selects DEP@UIS from database
18: CI sends PCA activation message to UIS referring to DEP@UIS
19: UIS determines the user pseudonym P_3 from DEP@UIS by Algorithm 24
20: UIS records AP activation transaction and μ under P_3
21: CI returns PCA activation conformation to AP
// AP transaction writing at TLP
22: AP uses PP to form EP@TLP by Algorithm 19
23: AP sends EP@TLP and AP activation transaction to TLP
24: TLP validates and decrypts EP by Algorithm 23
25: TLP records AP activation transaction under user pseudonym
26: AP securely deletes uPP // cleanup

We make some remarks on Protocol 7. As part of Line 3 also an intermediary CA certificate (DVCA) is sent residing between the TA certificate and the trust anchor (CVCA), see Appendix F. By its nature a PCA instance does not automatically uses an AP user registration process as in the regular setup, cf. Section 6.1. This particular hampers a help desk functionality at the AP. To allow for this, one could introduce a help desk register and include some additional steps in Protocol 7. In these steps the help desk register is sent some PCA instance data required for the help desk functionality together with data allowing an help desk employee to authenticate the user on the phone. This data could be dir-

ectly identifying information (name, date and place of birth and address) but could also be a nick same and some secret questions/answers. Note that the help desk register is not pseudonym based. To allow a help desk employee to view selections of authentication transaction with user consent an EP@TLP should also be recorded in the help desk register. As indicated in the remarks following Protocol 1 each query of a help desk employee should result in a log written to the TLP viewable by the user.

The reading of a randomized polymorphic form in Line 6 of Protocol 7 is also the essence of PCA authentication (Protocols 8, 9). As noted in Section 7 the AP bases its trust on the authenticity/integrity of the randomized polymorphic form read on the trust of the authenticity/integrity of EAC secure messaging setup in Line 4. This is similar to the setup of the German RI protocol, cf. Appendix F. By the privacy properties of EAC secure messaging this trust is not transferable to other parties. However, this trust can be transferred to the HSM at the PCA authentication provider by letting that manage both complete EAC tunneling and the transformation function. However, in EAC practice the HSM only manages the TA private key and returns the two secure messaging keys to the application communicating to the card. Alternatively, we could make the trust transferable by supplementing the PCA instance digitally sign the randomized polymorphic forms read by the authentication provider. By using the CA key for this (or or another shared key) this would not influence distinguishably. However this would have serious impact on both PCA computational and communicational performance. That is, in practice one needs to trust the PCA authentication provider not manipulating the polymorphic forms read before offering those to its HSM. In Section 8 we indicate that this only allows a fraudulent PCA authentication provider to provide rogue pseudonyms to service providers. The PCA authentication provider is not able to provide identities or pseudonyms to service providers of users of which has not acquired the corresponding polymorphic form for.

7.3 PCA Authentication

As with any polymorphic authentication, two types of PCA authentication exist: based on identity (BSN) and based on pseudonym. Both are specified in Protocols 8 and 9 respectively. In both protocols we assume that the PCA authentication provider is provided with the appropriate terminal certificates from the issuer and key material from the KMA. Compare the discussion prior to Protocol 7. We also assume that service providers are provided the secret EI/EP decryption and closing keys by the KMA, i.e. keys of type $\mathbf{ID_{Di}}, \mathbf{PD_{Di}}, \mathbf{PC_{Di}}$ and the scheme keys $\mathbf{Y}, \mathbf{Z}$ to verify EI/EP Schnorr signatures (cf. Table 6 of Appendix C.1).

Protocol 8 *PCA authentication providing identity*
User authenticates to Service Provider whereby providing his identity through PCA Authentication Provider.

1: `User wants to authenticate to SP providing his identity`

2: SP directs user to AP with request for identity based authentication
3: User enters PIN, allows eID client to establish local PACE SM with PCA
4: Authentication provider sends TA certificate to eID client
5: eID client arranges EAC SM between PCA and AP eID server
6: User consents to eID client on providing his identity to SP.
7: eID client enforces that AP eID server only reads randomized unsigned uPIP and uniqueness identifier μ from PCA
8: AP uses uPIP to form EP@SS by Algorithm 36 (Appendix D)
9: AP queries Status Service with EP@SS and μ
10: If query result is not ‘‘Activated’’ for the AP, then authentication Fails
11: AP uses uPIP to form EI for SP by Algorithm 35 (Appendix D)
12: User is redirected to SP with EI in response
13: SP validates and decrypts EI by Algorithm 21
14: On success, user is authenticated under his identity
15: On failure, user authentication at SP fails
// AP transaction writing at TLP
16: AP uses uPIP forms EP@TLP by Algorithm 36 (Appendix D)
17: AP sends EP@TLP and authentication transaction to TLP
18: TLP validates and decrypts EP by Algorithm 23
19: TLP records authentication transaction under user pseudonym
20: AP securely deletes uPIP // cleanup

Protocol 9 *PCA authentication providing pseudonym*
User authenticates to Service Provider whereby providing his (role based) pseudonym through PCA Authentication Provider.

1: User wants to authenticate to SP providing his pseudonym
2: SP directs user to AP with request for pseudonym based authentication
3: User enters PIN, allows eID client to establish local PACE SM with PCA
4: Authentication provider sends TA certificate to eID client
5: eID client arranges EAC SM between PCA and AP eID server
6: Optionally, the user chooses role in which to authenticate to SP
7: User consents to AP on providing (role based) pseudonym to SP
8: eID client enforces that AP eID server only reads randomized unsigned uPP and uniqueness identifier μ from PCA
9: AP uses uPP and forms EP@SS by Algorithm 37 (Appendix D)
10: AP queries Status Service with EP@SS and μ
11: If query result is not ‘‘Activated’’ for the AP, then authentication Fails
12: AP uses uPP and forms EP@SP by Algorithm 37 (Appendix D)
13: User is redirected to SP with EP in response
14: SP validates and decrypts EP by Algorithm 21
15: On success, user is authenticated under his pseudonym
16: On failure, user authentication at SP fails
// AP transaction writing at TLP
17: AP uses uPP of user and forms EP@TLP by Algorithm 37 (Appendix D)
18: AP sends EP@TLP and authentication transaction to TLP
19: TLP validates and decrypts EP by Algorithm 23
20: TLP records authentication transaction under user pseudonym
21: AP securely deletes uPP // cleanup

7.4 PCA Deactivation

In Protocol 10 we formalize deactivation of a user AP authenticator.

Protocol 10 *PCA card deactivation at AP*
PCA card deactivation at PCA authentication provider

1: User connects to the deactivation service of PCA authentication provider
2: User enters PIN, allows eID client to establish local PACE SM with PCA
3: Authentication provider sends TA certificate to eID client
4: eID client arranges EAC SM between PCA and AP eID server
5: eID client enforces that AP eID server only reads randomized usigned uPP and uniqueness identifier μ from PCA
6: AP uses uPP to form EP@CI by Algorithm 37 (Appendix D)
7: AP sends PCA deactivation message to CI referring to EP@CI and μ
8: CI determines the user pseudonym P_1 from EP by Algorithm 23
9: CI records AP deactivation using P_1, μ in database
// CI updates SS
10: CI selects DEP@SS from database
11: CI sends PCA deactivation message to SS referring to DEP@SS and μ
12: SS determines the user pseudonym P_2 from DEP by Algorithm 24
13: SS records AP deactivation transaction using user P_2, μ
// CI updates UIS
14: CI selects DEP@UIS from database
15: CI sends PCA activation message to UIS referring to DEP@UIS
16: UIS determines the user pseudonym P_3 from DEP@UIS by Algorithm 24
17: UIS records AP activation transaction using P_3, μ
18: CI returns PCA deactivation conformation to AP
// AP transaction writing at TLP
19: AP uses PP to form EP@TLP by Algorithm 19
20: AP sends EP@TLP and AP deactivation transaction to TLP
21: TLP validates and decrypts EP by Algorithm 23
22: TLP records AP deactivation transaction under user pseudonym
23: AP securely deletes uPP // cleanup

7.5 PCA Revocation

In Protocol 11 we formalize revocation of a user PCA authenticator at the card issuer.

Protocol 11 *User Authenticator Revocation*
User deactivates authenticator in scheme.

1: User requests CI for PCA revocation // e.g. using revocation password
// CI updates SS
2: CI selects DEP@SS and uniqueness identifier μ coupled to PCA card
3: CI registers card ‘‘Revoked’’ and μ to SS referring to DEP@SS
4: SS determines the user pseudonym P from DEP by Algorithm 24
5: SS registers card with uniqueness identifier μ ‘‘Revoked’’ under P
// CI updates UIS
6: CI selects DEP@UIS coupled to PCA card
7: CI registers card ‘‘Revoked’’ and μ to UIS referring to DEP

8: `UIS determines the user pseudonym` P' `from DEP by Algorithm 24`
9: `UIS registers card with uniqueness identifier ‘‘Revoked’’ using` P', μ

Proposition 7.1 *If we denote*

$$k = \lfloor \frac{Batch\_Size}{Usage\_Limit} \rfloor$$

then PCA authentications cryptographically provide k-anonymity sensu [47]. That is, authentications of a PCA instance cannot be cryptographically distinguished by the authentication provider to a group of holders of size less than k.

Proof: PCA instances are generated in batches of size Batch_Size. The fixed data on each instance, e.g. the CA certificate, readably by an authenticator coincides by construction. During PCA authentication an authorized authentication provider is provided with a uniqueness identifier μ and a randomized uPIP or uPP. The latter two are cryptographically indistinguishable by Proposition 4.2. So only the uniqueness identifier allows the authentication provider to distinguish PCA instances within the batch. Uniqueness identifiers are uniformly chosen in Protocol 26 implying that a batch falls into a number of Usage_Limit clusters each of minimal size k. It follows that there exist at least a k number of PCA instances in a batch with the same uniqueness identifier and the result follows. □

In Appendix G we motivate a choice of k-anonymity with k equal to 20.000. This can for instance be achieved using choosing Batch_Size = 100.000 and Usage_Limit = 5.

8 Comparison with requirements

8.1 eID reliability requirements

Below we compare the polymorphic eID scheme with the reliability requirements identified in Section 3.1.

R_1: Strong authentication With the exception of PCA the security of the means of authentication (authenticator) is not in scope of the polymorphic specification. In scope is the transport security of identifying information to relying parties (Service Providers). Compare the requirements of [19, Section 2.3.1]. These requirements stipulate that this transport shall be protected against replay and shall also protect the integrity and confidentiality of identifying information. This protection can be supplemented by the polymorphic setup, e.g. by using TLS, XML signatures and encryption. However, the polymorphic setup can be self-sustainable in this respect too. Indeed, in the polymorphic setup this identifying information takes the forms of EIs or (D)EPs. This information is ElGamal encrypted protecting the confidentiality of the plaintext payload. The encrypted data is also signed: a DEP

is ECDSA signed and an EI/EP is EC-Schnorr signed. Replay protection can be provided by incorporating a relying party specific random challenge in the EIs or (D)EPs. In the context of authentication, the service provider then includes such a challenge in the authentication request which is part of the scope of the EI/EP EC-Schnorr signature. See also Section H.1.

R_2: PCA design meets eIDAS level High The German eID card (nPA) has been peer reviewed by European member states to fulfill all requirements of eIDAS level of assurance High. Compare [20]. Like PCA, nPA allows to be used through an federative authentication provider (eID service). Compare Appendix F. Both are based on letting the AP setup EAC secure messaging with the card application and read data over it. In nPA the AP reads a complete service provider pseudonym ("RI") where in PCA the AP reads a randomized polymorphic form (PIP or PP). These are then formed into an identity or a service provider pseudonym using an HSM of the AP. Where nPA supports revocation through dedicated revocation lists, PCA supports this through its status service. This service also supports white listing which is only supported with nPA by distributing "white lists" when serious card security flaws occur. We conclude that if nPA design meets eIDAS level High then so does PCA design.

R_3: Independence of identities and pseudonyms This requirement is met by letting identities and pseudonyms be based on independent cryptographic infrastructures. The private EI decryption keys of service providers are based on the keys $\mathbf{y}$ and master key $\mathbf{IE}_{\mathrm{M}}$ whereas for EP these are based on the unrelated keys $\mathbf{z}$ master key $\mathbf{PE}_{\mathrm{M}}$.

R_4: BSN-L activation binding Encrypted Identity/Encrypted Pseudonyms are protected by an EC-Schnorr signature of which the private key resides in the HSM. That is, to generate an EI or EP an authentication provider is forced to use the relevant EI/EP generation function in the HSM (assuming the HSM is correctly secured). An HSM of a regular authentication providers only accepts polymorphic forms that are designated for the authentication provider and correctly signed by BSN-L. So clearly a regular authentication provider can only use PIP/PI/PP provided to him by BSN-L to produce legitimate EI/EP.

A PCA authentication provider uses unsigned PIPs and PPs. This in principle allows him to manipulate these before providing them to the EI/EP generating functions in the HSM.[2] We discuss the potential impact of this on the security of identities and pseudonyms delivered to service providers. With respect to the latter, a fraudulent PCA authentication provider could perform additional reshuffling operations on a PP. However, by doing so he can only provide rogue pseudonyms to service providers, i.e. pseudonyms that do not correspond to actual users (identities). Indeed, the keyed mapping in the payload of the PP ensure that the PP of another user is cryptographically unrelated. Furthermore, a PP of the user from another AP

[2] After Protocol 7 it is suggested letting PCA sign polymorphic forms with another shared key, but this hampers implementation too much.

is protected by the multiplication trapdoor using the key $\mathbf{AA_{Di}}$. Compare Line 4 of Algorithm 13 and Line 6 of Algorithm 14. The assumed hardness of the Diffie-Hellman problem prevents the AP to remove this key from the PP. We next discuss the impact on identities. The identities inside PIs are also protected by Optimal Asymmetric Encryption Padding (OAEP), cf. [3]. In Line 3 of Algorithm 12 and 14 a multiplication trapdoor using the key $\mathbf{AA_{Di}}$ is performed on an OAEP encoding P_1 of an identity *Id*. This leads to a point referred to as P_2. The trapdoor function brings us in the "Plaintext Awareness" setting of [3, Section 6]. Here an adversary game is considered where the adversary may ask such points P_2 for different identities and is challenged to generate one for an identity not asked. It is shown in [3] that an adversary that is able to win this game is also able to invert the trapdoor function. In our context an adversary that is able to generate an (unsigned) PI for an identity on basis of different PIs (other identities) is also able to win this game. Consequently such an adversary would be able to invert the multiplication trapdoor and thus to break the Diffie-Hellman problem in the surrounding group as observed in Section 4. The best strategy for a fraudulent authentication provider would be to try generating a unsigned PI as two randomly chosen elements from the group. By the OAEP properties the probability of success is less than 2^{-8h} where h is the hash length in bytes used in OAEP, cf. Section 4.7. In version 1 of the polymorphic scheme $h = 10$. Consequently this probability is less than 2^{-80} which is considered sufficiently small. Actually, in practice this probability is much smaller as the actual payload should consist of digits forming a BSN.
We conclude that a (PCA) authentication provider can only produce legitimate Encrypted Identities and/or Encrypted Pseudonyms by following the activation protocol through BSN-L.

$\mathbf{R_5}$: Authenticity of polymorphic forms The integrity and authenticity of DEPs, EIs and EPs can be verified by participants by checking the digital signature placed on them. Regular authentication providers can also verify the integrity and authenticity of PIP, PI and PP by checking the digital signature placed on them. PCA authentication providers cannot do so as the polymorphic forms read from an PCA instance are not signed (due to the randomization). However, these can infer the integrity and authenticity of the polymorphic forms in two ways. First they can infer it from the EAC tunnel of which the polymorphic forms are read. We note that in the German eID card (nPA) this is in fact is the only way of inferring integrity and authenticity of the pseudonym read from the card. In PCA the authentication providers can also infer integrity and authenticity from a successful result from the status service. This is particulary relevant when a CA key shared over a batch of PCA cards gets compromised. In the German eID card context the adversary can then create rogue cards corresponding with rogue pseudonyms. In the PCA context the adversary is not able to do so. He can only produce PCA cards for users of which has a copy of the polymorphic PCA card data. So unless the attacker also has access to a TA private key, breaks the TA protocol or gets this data from a compromised PCA authen-

tication provider he is not able to produce working PCA cards. Note that PCA authentication protocols specify that an authentication provider needs to delete the polymorphic forms read from the card after usage.

8.2 eID privacy requirements

P_1: Indistinguishability of PIs/PPs This requirement is met by the semantic security of the ElGamal encryption scheme, cf. Proposition 4.2.

P_2: PCA provides k-anonymity Compliance with this requirement is shown in Proposition 7.1.

P_3: PCA provides privacy friendly revocation Revocation can be done instantaneously by the PCA card issuer following Protocol 11. By revocation of a PCA instance only the status of the card at the PCA Status Service is affected, the service providers are not provided any information. That is, the pseudonymity of the user is unaffected.

P_4: Indistinguishability of EIs/EPs This is due to the semantic security of the ElGamal encryption scheme, cf. Proposition 4.2.

P_5: Indistinguishability of DEPs Compliance with this requirement follows from the semantic security of the ElGamal encryption scheme (Proposition 4.2) and Corollary 5.1. This corollary states that the pseudonym payload inside a DEP being is formed as a two party computation by BSN-L and the intended service provider.

P_6: Non-invertibility of pseudonyms Compliance with this requirement follows from Formula (19) which indicates that pseudonyms are derived from keyed hash values based on the identity.

P_7: Pseudonym unlinkability by SPs For space reasons we only provide a proof sketch which can be further formalized using standard arguments in the so-called random oracle model [2]. Consider two service providers SP_1, SP_2. As service providers themselves place the closing keys on pseudonyms, cf. Formula (19), we can disregard these for this requirement. That is, we are required to prove that service providers are not able to link their *unclosed* pseudonyms. One can actually prove this under two weaker conditions. We assume that the BSN-L part of the pseudonym is based on a publicly computable pseudo random function $\mathcal{I}(.)$ of Id, i.e. that the keys $\mathbf{IM_M}, \mathbf{IW_M}$ are known to the service providers. We also assume that $\mathrm{SP}_1, \mathrm{SP}_2$ know of two corresponding pseudonyms P_1, P_2 of a person with unknown identity Id. Now if we denote the expression $\mathcal{K}_1(\mathbf{PS_M}, R \| \mathrm{SP_i} \| \mathrm{KV})$ from Formula (19) by δ_i for $i = 1, 2$ then a unclosed pseudonym for $\mathrm{SP_i}$ takes the form $\delta_i \cdot \mathcal{I}(Id)$. If $\mathrm{SP}_1, \mathrm{SP}_2$ are able to decide if a SP_1 pseudonym of person with (unknown) personal numbers Id_a corresponds to a SP_2 pseudonym of person with (unknown) personal numbers Id_b then they able to decide if the quadruple

$$(\delta_1 \cdot \mathcal{I}(Id), \delta_1 \cdot \mathcal{I}(Id_a), \delta_2 \cdot \mathcal{I}(Id), \delta_2 \cdot \mathcal{I}(Id_b)),$$

is a DDH quadruple (cf. Section 4). We assumed that this problem was intractable.

P_8: Pseudonym unlinkability by SP and AP The proof of this requirement is similar to that of the previous requirement and we use its terminology. The crux is the secrecy of the closing key at SP_2. The proof can also be based on the weaker condition that the keys $\mathbf{IM_M}, \mathbf{IW_M}$ are known to the AP and SP_1 as well as δ_i for $i = 1, 2$. However, we no longer disregard the closing key $\mathbf{PC_{D2}}$. The pseudonyms at SP_2 take the form $\mathbf{PC_{D2}} \cdot \delta_2 \cdot \mathcal{I}(Id)$. Now, if SP_1 and an AP are able to decide if a SP_1 pseudonym of person with (unknown) personal numbers Id_a corresponds to a SP_2 pseudonym of person with (unknown) personal numbers Id_b then they able to decide if the quadruple

$$(\delta_1 \cdot \mathcal{I}(Id), \delta_1 \cdot \mathcal{I}(Id_a), \mathbf{PC_{D2}} \cdot \delta_2 \cdot \mathcal{I}(Id), \mathbf{PC_{D2}} \cdot \delta_2 \cdot \mathcal{I}(Id_b)),$$

is a DDH quadruple (cf. Section 4). We assumed that this problem was intractable.

P_9: Conformity of pseudonyms with legal principles The European General Data Protection Regulation [17] defines "pseudonymisation" as: *the processing of personal data in such a manner that the personal data can no longer be attributed to a specific data subject without the use of additional information, provided that such additional information is kept separately and is subject to technical and organisational measures to ensure that the personal data are not attributed to an identified or identifiable natural person.* The pseudonyms formed in the polymorphic setup adhere to this definition. Indeed, as indicated in Formula (19) a pseudonym is formed using cryptographic key material from BSN-K, authentication provider and the service provider. This key material is not only organisationally separated but also technically separated by the HSMs at BSN-L and the authentication providers. In 2007, the Dutch data protection authority (DPA) has issued a ruling [15] related to pseudonymisation by a trusted third party. In our context these technically stipulate that a) one should deploy good practice cryptographic techniques and BSN-L should use a one-way function, b) technical and organizational should prevent feasibility of brute forcing at the service provider. The service provider pseudonym produced in the polymorphic setup clearly meet these requirements.

8.3 eID usability requirements

U_1: Pseudonym compatibility Compliance with this requirement follows from Proposition 5.3.

U_2: Support for role base pseudonyms Roles can be introduced as part of DEP generation (Section 5.2.5) and as part of EP generation (Section 5.3.2). Effectively they are implemented as part of the service provider identity key derivation. Compare Formula (19).

9 Pseudonym conversion

In this section we specify two basic techniques for pseudonym conversion related to the following two use cases:

1. A service provider wants to change its closing key leading to new pseudonyms. See Algorithm 27. In this situation the service provider also wants to convert existing its pseudonyms in its existing registrations (i.e. those under the current closing key $\mathbf{PC_{Di}}$) to pseudonyms under the new closing key $\mathbf{PC_{Di}}'$. This process can be triggered by a technical change of the closing key but also by a functional desire of the service provider to periodically change its pseudonyms for security and privacy reasons.
2. A service provider SP_i wants to exchange pseudonymous information with another service provider SP_j. See Protocol 12. Here SP_i needs to convert its existing pseudonyms to those of service provider SP_j. Such conversions also need to arrange for the situation that service providers use role based pseudonyms, whereby the providers use different role indicators (diversifiers). A particular situation arises when one service provides wants to combine different role based pseudonyms. As such conversions are considered security and privacy sensitive operations as these are in clear contrast with the pseudonym unlinkability requirements. As will be come apparent in Protocol 12, such conversions require SP_i, SP_j to receive conversion keys from the KMA. We assume that the KMA has processes in place on handling requests for such conversion keys whereby legitimacy is also reviewed. Valid legitimacy reasons could include a service provider changing its name from SP_i to SP_j and service provider SP_i merging with service provider SP_j.

Algorithm 27 corresponds to the first use case mentioned above. Prior to running Algorithm 27 the service provider determines a pseudonym conversion key

$$\Gamma = \frac{\mathbf{PC_{Di}}'}{\mathbf{PC_{Di}}}.$$

Algorithm 27 Service provider pseudonym conversion due to closing key change
Conversion of SP_i pseudonym P under closing key $\mathbf{PC_{Di}}$ to closing key $\mathbf{PC_{Di}}'$ based on pseudonym conversion key Γ.

1: `Retrieve` Γ.
2: `Return` $P' = \Gamma \cdot P$.

The correctness proof of Algorithm 27 simply follows by comparison with Equation (19). For simplicity Algorithm 27 above only converts one pseudonym converted to a new one. One can easily extend this to a list of pseudonyms. In the protocol below two service providers are introduced denoted by SP_i and SP_j. We let $\mathrm{SP_{IDi}}, \mathrm{SP_{IDj}}$ denote the name identifer of the respective service providers. Furthermore $\mathbf{PC_{Di}}, \mathbf{PC_{Dj}}$ denote the closing keys of respective service providers.

Protocol 12 Pseudonymous information exchange between service providers.
Conversion of (P_1, I) *pertaing a* SP_i *pseudonym* P_1 *under role* R_1 *and coupled information* I *to a* SP_j *pseudonym* P_2 *under role* R_2.

1: SP_i, SP_j request the KMA for conversion keys // legitimacy checked
2: If request is not honored by KMA protocol ends
3: KMA generates $t \in_R \mathbb{F}_q^*$ as conversion key and sends this to SP_j.
4: KMA generates Γ as in formula below and sends this to SP_i

$$\Gamma = \frac{\mathbf{PC_{Dj}} \cdot \mathcal{K}_1(\mathbf{PS_M}, R_1 \,|||\, \mathrm{SP_{IDj}} \,|||\, \mathrm{KV})}{t \cdot \mathbf{PC_{Di}} \cdot \mathcal{K}_1(\mathbf{PS_M}, R_2 \,|||\, \mathrm{SP_{IDi}} \,|||\, \mathrm{KV})} \tag{25}$$

5: SP_i calculates $P' = \Gamma \cdot P_1$
6: SP_i sends P' and I to SP_j
7: SP_j calculates $P_2 = t \cdot P'$ and stores I under P_2

In the situation that SP_i and SP_j are one organisation entity, e.g. when SP_j is a new name for SP_i or when SP_i and SP_j merge into SP_j, this property is not relevant. One can then discard key t in Protocol 12, i.e. this is effectively accomplished by taking $t = 1$ in Step 3 of Protocol 12. In this way, service provider SP_i directly outputs the pseudonyms of SP_j in Step 5 of Protocol 12. If one does not want to make a difference in key generation for the KMA in this situation, the service provider can also compute $t' = \Gamma \cdot t$ and replace Steps 5-7 in Protocol 12 with the computation of $P_2 = t' \cdot P_1$. This is more efficient as it saves one Elliptic Curve computation. Note that in our notation the operation "$\cdot$" is a multiplication in the field $\mathbb{F}_q$, in practice this means a multiplication of two integers, i.e. Γ and t, followed by a modulo q operation.

The following Algorithms 28 and 29 are a further application of this technique in the situation that service providers SP_i and SP_j are one organisation entity. In these algorithms we assume that the entity is provided the keys t, Γ from the KMA as indicated in 12. The input of Algorithm 28 is an SP_i encrypted pseudonym under role R_1 and the output is an SP_j pseudonym P_2 under role R_2. Algorithm 29 is similar but takes an direct encrypted pseudonym as input. These algorithms are quite straightforward but are presented here for completeness reasons.

Algorithm 28 *DecEPC*(EP)
Validated generation of SP_j pseudonym under role R_2 by service provider SP_i based on Encrypted Pseudonym $\mathrm{EP} = (\mathrm{EP_{ID}}, R, A, B, C, D)$ under role R_1.

1: Validate $(\mathrm{EP_{ID}}, R, A, B, C, D)$ is correctly formed and not expired, on failure return Error // input validation, assessment of $C = \tilde{T}$
2: Interpret A as 3-tuple $(A_1, A_2, \mathbf{PD_{Pi}}) \in \mathbb{G}^3$ on failure return Error // parsing of EP
3: If $Ver_{\mathrm{Schn}}(\mathrm{EP_{ID}} || R || A || B || C, D, \mathbf{PD_{Pi}}, \mathbf{Z})$=False return Error // signature check
4: Look up key $\mathbf{PC_{Di}}$ compatible with EP, on failure, return Error
5: Compute $t' = \mathbf{PC_{Di}} \cdot t \cdot \Gamma$
6: Compute $(B_1, B_2, \mathbf{PD_{Pi}}) = \mathcal{RS}(A_1, A_2, \mathbf{PD_{Pi}}, t')$ // pseudonym closing

7: Look up key $\mathbf{PD_{Di}}$ corresponding with $\mathbf{PD_{Pi}}$
8: Compute $P = \mathcal{EG}_d(B_1, B_2, \mathbf{PD_{Pi}}, \mathbf{PD_{Di}})$ // ElGamal decryption (Section 4.2)
9: Return $(\mathtt{P_{ID}}, P)$

Algorithm 29 *DecDEPC*(DEP)
Validated generation of SP_j pseudonym under role R_1 by service provider SP_i based on Direct Encrypted Pseudonym $\mathrm{DEP} = (\mathrm{DEP_{ID}}, \mathrm{SP_{IDr}}, R, A, B, C, D)$ for role R_2.

1: Validate $(\mathtt{DEP_{ID}}, \mathtt{SP_{IDr}}, R, A, B, C, D)$ is correctly formed and not expired, on failure return Error // input validation, assessment of $C = \tilde{T}$
2: Interpret A as 3-tuple $(A_1, A_2, \mathbf{PD_{Pi}}) \in \mathbb{G}^3$ on failure return Error // parsing of DEP
3: If $Ver_{\mathrm{dsa}}(\mathtt{DEP_{ID}} || \mathtt{SP_{IDr}} || R || A || B || C, D, \mathbf{U})$=False return Error // signature check
4: Look up key $\mathbf{PC_{Di}}$ compatible with DEP, on failure, return Error
5: Compute $t' = \mathbf{DR_{Di,R}}.\mathtt{Keyd[0]} \cdot \mathbf{PC_{Di}} \cdot t \cdot \Gamma$
6: Compute $(E_1, E_2, \mathbf{PD_{Pi}}) = \mathcal{RS}(D_1, D_2, \mathbf{PD_{Pi}}, t')$ // pseudonym closing
7: Look up key $\mathbf{PD_{Di}}$ corresponding with $\mathbf{PD_{Pi}}$
8: Compute $P = \mathcal{EG}_d(E_1, E_2, \mathbf{PD_{Pi}}, \mathbf{PD_{Di}})$ // ElGamal decryption (Section 4.2)
9: Return $(\mathtt{P_{ID}}, P)$

10 Key roll-over

In line with [38] a cryptographic key can have the following states: pre-active, active, suspended, deactivated, compromised and destroyed. In this section we focus on pre-active, active and deactivated states of keys in the polymorphic eID scheme. When a key is deactivated, one can opt for its destruction but this is not addressed in this document.

In principle, multiple of the polymorphic eID scheme key types, cf. Table 6, can be active at the same time. The basic rule is that parties are required to use the most actual keys (technically) available. In this section we describe the (technical) operations allowing parties to migrate from one key to a next one and to start using the new key. This process is called key *rollover*. In addition to this, we formulate suggestions for the maximal periods after which keys should be rolled over and deactivated. Recall that we specified in Section 4.9 that keys have an indication of their generation and activation times. Also, cryptograms have an indication of their generation times. This technically supports that parties can decide if a certain operation is allowed or not based on these times. Many of the keys in the scheme are managed in an HSM allowing technical enforcement of key deactivation.

In Table 3 below we have placed all cryptographic keys used in the polymorphic eID scheme from Table 6 into six groups. For readability, the first five columns of Tables 3 and 6 coincide. The group number is indicated by the last

column in Table 3. For each of the six keygroups we define a specific roll-over procedure, see Sections 10.1 - 10.6 below. The six roll-over periods are respectively denoted by the parameters S, E, I, R, P, M and Ψ respectively. Compare the sixth column of Table 3. The keygroups are organized in such a way that rolling over is increasingly hard, i.e. rolling over to a new Keygroup #1 is easiest and to a new Keygroup #6 hardest. The roll-over to a new Keygroup #6 is implemented by rolling to all new keys and/or a new cryptographic group $\mathbb{G} = (\langle G \rangle, +)$ which is addressed in Section 10.7. The latter typically relates to choosing a new elliptic curve, e.g. replacing the Brainpool320r1 curve to the Brainpool384r1 or Brainpool512r1 curve.

The seventh column of Table 3 indicates deactivation of the keygroup, i.e. the moment in time the keys are no long used in production. This column uses the additional six parameters C, M, L_{EI}, L_{EP}, L_{DEP}, L_{DEI} and Ω relating to the usage period of certain cryptograms indicated in Table 2. One does need to deploy all seven parameters, e.g. one could let $L_{\text{EI}}, L_{\text{EP}}$, $L_{\text{DEP}}, L_{\text{DEI}}$ and M, Ω coincide. But separate notation used for completeness sake.

Parameter	Meaning
C	Maximum validity period of a polymorphic form, typically the validity period of an PCA card, e.g. 10 years. Compare Section 10.1.
M	Pseudonym migration period in which service providers, have to migrate to new keys leading to different pseudonyms, e.g. 2 years. Compare Section 10.5.
Ω	Pseudonym migration period in which service providers, have to migrate to a new cryptographic group (elliptic curve) leading to different pseudonyms, e.g. 2 years. Compare Section 10.6.
L_{EI}	Lifetime of an Encrypted Identity, typically seconds but can be substantially longer to support caching.
L_{EP}	Lifetime of an Encrypted Pseudonym, typically seconds but can be substantially longer to support caching.
L_{DEI}	Lifetime of a Direct Encrypted Pseudonyms, can be seconds but is typically substantially longer to support caching.
L_{DEP}	Lifetime of a Direct Encrypted Identity, can be seconds but is typically substantially longer to support caching.

Table 2. De-activation parameters

It is vital that all parties in the scheme can reliably assess if new keys become (in-)active. To this end, we assume that a reliable, central repository exists in the eID scheme allowing changes in cryptographic keys to be communicated to all parties in the scheme. This is typically known as the *scheme metadata* and this is also the term we use. The scheme metadata is signed data allowing parties to obtain security relevant data of parties that participate in the scheme. This includes which keys are issued to/available for parties, assess their age and their NIST status (pre-activated, activated, suspended, deactivated, compromised). Parties are required to periodically, e.g. hourly, to look for key related changes in the metadata and to implement these changes in their systems. Coupled to each key registration are its relevant key metadata, i.e. the fields described in

Section 4.9.1. This includes the key version and the key generation date/time. In case the key type is public key, the key itself can also be registered into the metadata. To communicate a upcoming key activation, the key is generated in advance and its metadata is placed in the metadata with an activation time in the (near) feature. By actually delivering the keys to the involved parties well in advance of the actual activation time, parties have ample time implementing the new keys. Before their activation, these keys then have pre-activated status. The metadata can also be used to communicate an imminent deactivation time of keys. A certain period after the key is deactivated, the key reference could also be removed from the metadata. We note that in case the key is of public key type its metadata could take the form of a X.509 certificate [13]. However we do not require this.

#	ID	Name	Type	Function	roll-over Period (years)	Deactivation Period (years)	Section	Key-group
12.	**u**	PI/PP Signing key	EC Private Key (ECDSA)	Signing of PI, PP DEP and DEI	S:=3	S=3	10.1	1
13.	**U**	PI/PP Verification key	EC Public Key (ECDSA)	Verifying of PI, PP DEP and DEI		S+C=13		
21.	$\mathbf{ID_{Di}}$	Identity Decryption Derived key	EC Private key (ElGamal)	EI to I decryption	E:=3	$E = 3$	10.2	2
22.	$\mathbf{ID_{Pi}}$	Identity Decryption Public key	EC Public key (ElGamal)	EI validation (EC-SCHNORR)		E+Max($L_{\mathrm{EI}}, L_{\mathrm{DEI}}$)		
23.	$\mathbf{PD_{Di}}$	Pseudonym Decryption Derived key	EC Private key (ElGamal)	EP to P decryption		$E = 3$		
24.	$\mathbf{PD_{Pi}}$	Pseudonym Decryption Public key	EC Public key (ElGamal)	EP validation (EC-SCHNORR)		E+L_{EP}		
11.	$\mathbf{DT_{Di,R}}$	Direct Transmission Derived key	EC Private key (Diffie-Hellman) EC Public Key (ElGamal)	DEP generation by BSN-L		$E = 3$		
20.	$\mathbf{DR_{Di,R}}$	Direct Receiving Derived key	EC Private key (Diffie-Hellman) EC Private Key (ElGamal)	DEP to EP (special SPs only)		E+L_{DEP}		
14.	$\mathbf{IE_M}$	Identity Encryption Master key	HMAC key (Master key)	PI to EI transform	I:=5	I+E=8	10.3	3
15.	$\mathbf{IE_{Di}}$	Identity Encryption Derived key(ephemeral)	EC Private key (Diffie-Hellman)	PI to EI transform		-		
16.	$\mathbf{PE_M}$	Pseudonym Encryption Master key	HMAC key (Master key)	PP to EP transform		I+E=8		
17.	$\mathbf{PE_{Di}}$	Pseudonym Encryption Derived key(ephemeral)	EC Private key (Diffie-Hellman)	PP to EP transform		-		
6.	$\mathbf{DC_M}$	Direct Communication Master Key	HMAC key (Master key)	DEP deployment (special SPs only)		I=5		
27.	$\mathbf{SED_t}$	Supervisor Encryption Derived key	AES key	Auditelement decryption		-		
1.	**y** (or $\mathbf{IP_M}$)	Identity Private Master key	EC Private Key (ElGamal)	Generation of SP EI decryption keys	R:=8	R=8	10.4	4
2.	**Y** (or $\mathbf{IP_P}$)	Identity Private Public key	EC Public Key (ElGamal)	PI generation				
3.	**z** (or $\mathbf{PP_M}$)	Pseudonym Private Master key	EC Private Key (ElGamal)	Generation of SP EP decryption keys				
4.	**Z** (or $\mathbf{PP_P}$)	Pseudonym Private Public key	EC Public Key (ElGamal)	PP generation				
9.	$\mathbf{AA_M}$	Authentication provider Adherence Master key	HMAC key (Master key)	PP/PI generation (make AP specific)				
26.	$\mathbf{SED_a}$	Supervisor Encryption Derived key	AES key	Auditelement decryption		-		
10.	$\mathbf{AA_{Di}}$	Authentication provider Adherence Derived key	HMAC key (Master key)	Polymorphic transform		R+C=18		
5.	$\mathbf{PC_M}$	Pseudonym Closing Master key	HMAC key (Master key)	Generation of SP closing keys	P:=10	P=10	10.5	5
18.	$\mathbf{PS_M}$	Pseudonym Shuffle Master key	HMAC key (Master key)	PP to EP transform		P+M=12		
19.	$\mathbf{PS_{Di,R}}$	Pseudonym Shuffle Derived key(ephemeral)	EC Private key (Diffie-Hellman)	PP to EP transform		-		
25.	$\mathbf{PC_{Di}}$	Pseudonym Closing Derived key	EC Private key (Diffie-Hellman)	EP to P decryption				
7.	$\mathbf{IW_M}$	Identity Wrapping Master key	HMAC key (Master key)	PP generation (map into curve)	Ψ	Ω	10.6	6
8.	$\mathbf{IM_M}$	Identity Mapping Master key	HMAC key (Master key)	PP generation (map inside curve)				

Table 3. eID (Scheme) Keys

10.1 Roll-over for Keygroup #1

Every S years BSN-L generates a new public key pair u, U and stores U in the metadata coupled with its key fields including key versions, i.e. the key version sequence KVS, and generation time. After a certain communication time all parties using polymorphic and encrypted forms, i.e. authentication providers and service providers, are required to have implemented the new verification public key U in their systems. After this communication time, BSN-L starts using the new private key u to sign polymorphic forms (PIP, PI, PP) and encrypted forms (DEI, DEP). Parties consuming polymorphic or encrypted forms can then verify these by using the published public key U.

Key deactivation time
After BSN-L starts using the new signing key u the previous signing key u can be deactivated and in fact destroyed. That is, the deactivation time of u corresponds with S. The public key U needs to be active longer to verify active polymorphic or (direct) encrypted forms signed with u. The polymorphic forms, most notably the ones residing in a PCA card will likely to be used longest. See Section 7. We remark that the polymorphic structures (PI, PP) reside in unsigned form in a PCA card and the public key U plays no role in the actual usage of the PCA card. One can thus argue that PCA lifetime is only of minor relevance for the public key U active period. However, this relevance could be based on auditing requirements. If we denote the maximum validity of a polymorphic form (PCA card) by C years then the minimum deactivation time of a public key U is after $S+C$ years. We suggest using this minimum deactivation time.

Suggestions on roll-over and deactivation time periods
The key pair u, U resembles both the country signer and document signer key in the context of electronic passports, cf. [26]. Based on that resemblance and part 12 of [26], a roll-over period S between 3 month and 5 years seems appropriate. We suggest a roll-over period of 3 years, i.e. $S = 3$. In the Netherlands identity documents have a maximum validity of 10 years so this appears to be a suitable choice for C. This means that the minimum deactivation time of a public key U is 13 years, which we suggest using.

10.2 Roll-over for Keygroup #2

Public keys of type $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ in Keygroup #2 are made available in the scheme metadata coupled with their key fields including key versions and activation time. There can be more than one of such keys in the metadata. As part of the transformation operation at the authentication provider, a service provider refers to a key of type $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ including its key versions. Compare Section 5.3. The authentication provider then inspects the metadata to see if the key is acceptable, i.e. exists and is valid. If and only if these conditions are satisfied, the authentication provider runs one of the appropriate Algorithms 17, 18, 19, 20 which automatically forms an encrypted form corresponding to the

referred key. Within E years after activation, the keys $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ will need to be renewed. Service providers need to request new keys from the KMA and the public parts will be placed in the metadata as indicated above. From then on, service providers can refer and use the new keys $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$.

Keys of type $\mathbf{DT_{Di,R}}, \mathbf{DR_{Di,R}}$ in Keygroup #2 are generated by the KMA and provided to BSN-L and the related service provider. Their key fields including key versions and generation time will be placed in the scheme metadata. As part of the generation of a direct encrypted form, compare Algorithms 16, 15, the calling party needs to refer to keys of type $\mathbf{DT_{Di,R}}, \mathbf{DR_{Di,R}}$. Apart from verifying whether the calling party is authorized to request the direct encrypted forms, BSN-L also inspects the metadata to see if the keys exists, are valid. Within E years after activation, the keys $\mathbf{DT_{Di,R}}, \mathbf{DR_{Di,R}}$ will need to be renewed. From then on the relying parties can refer and use the new keys $\mathbf{DT_{Di,R}}, \mathbf{DR_{Di,R}}$.

Key deactivation time
If the lifetime of encrypted forms (EI, EP, DEI, DEP) would be in the order of seconds, then the deactivation period of the keys in Keygroup #2 can also be set to E. However, if encrypted form caching is allowed then these lifetimes can be substantially longer than seconds. In that case the deactivation period of the keys in Keygroup #2 needs to reflect this. This is indicated in the seventh column of Table 3.

Suggestions on roll-over and deactivation time periods
Keys in Keygroup #2 are used within a private community namely the parties involved the eID scheme. The keys resemble thus the public/private key pairs for end-user organisations indicated in part g of the Dutch government PKI requirements, cf. [43]. These requirements stipulate a maximum validity period of 5 years. This implies a maximal validity period of a key in Keygroup #2 of 5 years We suggest a roll-over period of 3 years, i.e. $E = 3$. We do not make suggestions on deactivation times for Keygroup #2 as this strongly coupled with caching use cases of encrypted forms, e.g. "recurrent activation" as indicated in Section 5.2.4.

10.3 Roll-over for Keygroup #3

Keys of type $\mathbf{IE_M}$ are generated by the KMA and used together with keys of type $\mathbf{y}$ to generate the private keys of type $\mathbf{ID_{Di}}$ for service providers. Similarly, keys of type $\mathbf{PE_M}$ are generated by the KMA and used together with keys of type $\mathbf{z}$ to generate keys of type $\mathbf{PD_{Di}}$ for service providers. Compare Table 7. The KMA will always use the most recent version of the keys $\mathbf{IE_M}, \mathbf{PE_M}, \mathbf{y}, \mathbf{z}$ available to generate keys of type $\mathbf{ID_{Di}}, \mathbf{PD_{Di}}$ and their public parts $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$. Also, the KMA makes keys $\mathbf{IE_M}, \mathbf{PE_M}$ available in the HSMs of authentication providers coupled with their key fields including key versions and activation time.

Within I years after activation, the KMA generates new keys $\mathbf{IE_M}$, $\mathbf{PE_M}$, $\mathbf{DC_M}$ and announces their existence in the scheme metadata including their key versions and activation time. The KMA will also make the keys $\mathbf{IE_M}, \mathbf{PE_M}$

available in the authentication provider HSMs coupled with their key fields including key versions and activation time. When the activation time is reached, both the KMA and authentication providers start using the keys. The KMA will produce new keys of type $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ for service providers which will be registered in the scheme metadata.[3] With the possession of the keys $\mathbf{IE_M}, \mathbf{PE_M}$ in their HSMs, the authentication providers can support these new keys as part of the generation of EI/EP encrypted forms, i.e. in Algorithms 17, 18, 19, 20. From this moment on, the authentication providers will also be able to use a new version of the key $\mathbf{SED_t}$ that is based on the key $\mathbf{PE_M}$.

The KMA will also the new key $\mathbf{IE_M}$ in the generation and issuance of the key of type $\mathbf{ID_{Pi}}$ to BSN-L honoring a DEI service application, cf. Section 5.2.4. Likewise, the KMA will use the new keys $\mathbf{PE_M}$ and $\mathbf{DC_M}$ key in honoring a DEP service application, cf. Section 5.2.5. This will lead to generating new keys of type $\mathbf{DT_{Di,R}}$ for BSN-L and corresponding keys of type $\mathbf{DR_{Di,R}}$ for service providers. After activation (e.g. indicated in the scheme metadata), BSN-L will issue DEIs/DEPs based on the new keys. That is, BSN-L will use most recent (active) keys in its generation of DEIs/DEPs.

As indicated in Section 10.2 as part of the transformation request, a service provider refers to a key of type $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ including the key versions of the keys therein. See Section 4.9.1. Based of the KVS indicated, the authentication provider will use appropriate versions of the keys $\mathbf{IE_M}, \mathbf{PE_M}$ and $\mathbf{SED_t}$.

Key deactivation time
After a new key of type $\mathbf{DC_M}$ is available the previous version is no longer used. This implies that the deactivation time of keys of type $\mathbf{DC_M}$ can be set to the activation time, i.e. I. In principle, keys of type $\mathbf{IE_M}, \mathbf{PE_M}$ can be used together with public keys of type $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ based on $\mathbf{IE_M}, \mathbf{PE_M}$ in a period of E years after their generation. This implies that the deactivation time period of the keys $\mathbf{IE_M}, \mathbf{PE_M}$ can be set to I+E, which we suggest.

Suggestions on roll-over and deactivation time periods
Keys of type $\mathbf{IE_M}, \mathbf{PE_M}$ resemble the private signing key of an intermediary Certification Authority. Dutch government requirements, cf. [44], stipulate a maximal validity period of 12 years for such keys. We suggest a validity (deactivation time) period (= I+E) of 8 years. Given that $E = 3$, this implies a roll-over period I of 5 years.

10.4 Roll-over for Keygroup #4

Keys of type $\mathbf{y}$ are generated by the KMA and used together with keys of type $\mathbf{IE_M}$ to generate keys of type $\mathbf{ID_{Di}}$ for service providers. Its public counterpart $\mathbf{Y}$ is used by BSN-L in the generation of polymorphic identities (PI and PIP).

[3] In the $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ key derivation algorithm we have ensured that sets of $\mathbf{IE_M}, \mathbf{PE_M}$ keys can be securely used together with different sets of keys $\mathbf{y}, \mathbf{z}$. See Appendix C.2.

Similarly, keys of type $\mathbf{z}$ are generated by the KMA and used together with keys of type $\mathbf{PE_M}$ to generate keys of type $\mathbf{PD_{Di}}$ for service providers. The public counterpart $\mathbf{Z}$ is used in the generation of polymorphic pseudonyms (PP and PIP). The keys of type $\mathbf{z}$ are also used to generate the "$\mathbf{PD_{Di}}$" part (i.e. Keyd[1]) of the keys of type $\mathbf{DR_{Di,R}}$, cf. Appendix C.2.

Within R years after the previous activation, the KMA generates new keys of type $\mathbf{y}$, $\mathbf{Y}$, $\mathbf{z}$, $\mathbf{Z}$, $\mathbf{AA_M}$ and announces their existence in the scheme metadata including their key versions and activation time.

When the activation time is reached, both the KMA and BSN-L start using the new keys and do no longer use the previous keys. The KMA will use them all to generate new service provider public/private keys. BSN-L will use the new keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$ in the generation of new polymorphic forms (PIP, PI, PP) for authentication providers. This means that after activation, four possibilities can occur in a transformation request from a service provider to an authentication provider. This is indicated in the rows 1-4 of Table 4 below. Rows 5-8 are only placed for illustrative reasons, indicating the combination with roll-over of Keygroup #4 specified in Section 10.4.

#	PIP/PI/PP at AP based on	SP public keys $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ based on	SP public keys $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ based on
1.	Initial keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	Initial keys of type $\mathbf{y}$, $\mathbf{z}$	Initial keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
2.	New keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	Initial keys of type $\mathbf{y}$, $\mathbf{z}$	Initial keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
3.	Initial keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	New keys of type $\mathbf{y}$, $\mathbf{z}$	Initial keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
4.	New keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	New keys of type $\mathbf{y}$, $\mathbf{z}$	Initial keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
5.	Initial keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	Initial keys of type $\mathbf{y}$, $\mathbf{z}$	New keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
6.	New keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	Initial keys of type $\mathbf{y}$, $\mathbf{z}$	New keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
7.	Initial keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	New keys of type $\mathbf{y}$, $\mathbf{z}$	New keys of type $\mathbf{IE_M}, \mathbf{PE_M}$
8.	New keys of type $\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{AA_M}$	New keys of type $\mathbf{y}$, $\mathbf{z}$	New keys of type $\mathbf{IE_M}, \mathbf{PE_M}$

Table 4. Possible cases in Keygroup #4 roll-over (rows 1-4)

In none of these cases we want the service provider to (explicitly) perform migration activities. That is, we want the service provider receiving a response according to the appropriate Algorithms 17, 18, 19, 20 and his public key $\mathbf{ID_{Pi}}$, or $\mathbf{PD_{Pi}}$. In the first and fourth case the authentication provider can simply accomplish this by using the algorithms referred to. In the second and third case we support that the authentication provider can map the available polymorphic form to one matching the $\mathbf{ID_{Pi}}$, or $\mathbf{PD_{Pi}}$. This allows the authentication provider to subsequently use the appropriate Algorithms 17, 18, 19, 20.

To this end, we let $\mathbf{y}[i], \mathbf{Y}[i], \mathbf{Z}[i], \mathbf{z}[i], \mathbf{AA_{Di}}[i]$ denote the i-th version of the referred keys with $i = 1, 2, \ldots, n$ where n is the latest version. For $1 \leq i, j \leq n$ we define

$$\Delta y_{i,j} = \frac{\mathbf{y}[j]}{\mathbf{y}[i]}; \Delta z_{i,j} = \frac{\mathbf{z}[j]}{\mathbf{z}[i]}. \tag{26}$$

Notice that $\Delta y_{i,j} = \Delta y_{j,i}^{-1}$ and that $\Delta y_{i,i} = 1$. We remark that keys of type $\Delta y_{i,j}$, $\Delta y_{i,j}$ do not precisely fit the setup of Section 4.9.1 as they are based on two versions of the keys of $\mathbf{y}$ and $\mathbf{z}$. In other words, the value of the $\mathbf{y}, \mathbf{z}$ entry in the KVS of $\Delta y_{i,j}$, $\Delta y_{i,j}$ is unspecified. Although these KVS do not play a role in our constructions, for completeness we let the KVS entry at $\mathbf{y}$ of $\Delta y_{i,j}$ hold both $\mathbf{y}[i].KV$ and $\mathbf{y}[j].KV$ Similarly we let the KVS entry at $\mathbf{z}$ of $\Delta z_{i,j}$ hold $\mathbf{z}[i].KV$ and $\mathbf{z}[j].KV$. A simple construction is letting these entries be equal to $256 \cdot \mathbf{y}[j].KV + \mathbf{y}[i].KV$ and $256 \cdot \mathbf{z}[j].KV + \mathbf{z}[i].KV$ respectively. Which works assuming that the $\mathbf{y}$, $\mathbf{z}$ versions are integers less than 256.

Algorithm 30 *PI-Match*(PI, $\mathbf{ID_{Pi}}$) Transforming a polymorphic identity PI to match the Keygroup #4 version of $\mathbf{ID_{Pi}}$

1: Validate that PI is correctly formed as $(\mathtt{PI_M}, \mathtt{uPI}, AB, \tilde{T}, Sig)$ and not expired, on failure return Error // input validation, assessment of $\tilde{T}$
2: Represent uPI, $AB, \tilde{T}$ as byte array B // e.g. DER encoding
3: If $Ver_{\mathrm{dsa}}(B, Sig, \mathbf{U})$=False return Error // signature check
4: Interpret uPI as $(\mathtt{uPI_M}, A)$ with A an 3-tuple $(A_1, A_2, \mathbf{Y}_i) \in \mathbb{G}^3$ on failure return Error // parsing, uPI based on Keygroup #4 version i
5: Determine j, i.e. the Keygroup #4 version on which $\mathbf{ID_{Pi}}$ is based
6: If $i = j$ return uPI // already matching
7: Look up keys $\mathbf{AA_{Di}}[i]$, $\mathbf{AA_{Di}}[j]$, on failure, return Error
8: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}}[i] \cdot \mathbf{AA_{Di}}[j]^{-1})$ // transform to $\mathbf{AA_{Di}}_j$
9: Compute $E_3 = \mathcal{RK}(E_2, \Delta y_{i,j})$ // rekey to Y_j
10: Wrap E_3 in unsigned PI cryptogram and return this

Algorithm 31 *PP-Match*(PP, $\mathbf{PD_{Pi}}$) Transforming a polymorphic pseudonym PP to match the Keygroup #4 version of $\mathbf{PD_{Pi}}$

1: Validate that PP is correctly formed as $(\mathtt{PP_M}, \mathtt{uPP}, AB, \tilde{T}, Sig)$ and not expired, on failure return Error // input validation, assessment of $\tilde{T}$
2: Represent uPP, $AB, \tilde{T}$ as byte array B // e.g. DER encoding
3: If $Ver_{\mathrm{dsa}}(B, Sig, \mathbf{U})$=False return Error // signature check
4: Interpret uPP as $(\mathtt{uPP_M}, A)$ with A an 3-tuple $(A_1, A_2, \mathbf{Z}_i) \in \mathbb{G}^3$ on failure return Error // parsing, uPP based on Keygroup #4 version i
5: Determine j, i.e. the Keygroup #4 version on which $\mathbf{PD_{Pi}}$ is based
6: If $i = j$ return uPP // already matching
7: Look up keys $\mathbf{AA_{Di}}[i]$, $\mathbf{AA_{Di}}[j]$, on failure, return Error
8: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}}[i] \cdot \mathbf{AA_{Di}}[j]^{-1})$ // transform to $\mathbf{AA_{Di}}_j$
9: Compute $E_3 = \mathcal{RK}(E_2, \Delta z_{i,j})$ // rekey to Z_j
10: Wrap E_3 in unsigned PP cryptogram and return this

These algorithms are designed such that they conveniently fit in the existing transformation algorithms:

- by replacing lines 2-5 in Algorithm 17 with Algorithm 30 one obtains an algorithm transforming a Polymorphic Identity into an Encrypted Identity matching the Keygroup #4 version of the service provider public key $\mathbf{ID_{Pi}}$.

- by replacing lines 2-5 in Algorithm 19 with Algorithm 31 one obtains an algorithm transforming a Polymorphic Identity into an Encrypted Identity matching the Keygroup #4 version of the service provider public key $\mathbf{PD_{Pi}}$.

The described techniques only to relate to roll-over of Keygroup #4. However they can be easily combined with roll-over of Keygroup #3 too as Algorithms 30, 31 are independent of the Keygroup #3 version used. These use cases correspond with rows 5-8 of Table 4. In these cases one simply uses the Keygroup #3 version in EP/EP generation Algorithms 17, 19 indicated in the service provider public key.

The correctness of Algorithms 30, 31 is straightforward. One can easily combine these algorithms leading to a similar algorithm handling PIPs (Polymorphic Identity and Pseudonyms). As an illustration, we show the correctness of Algorithm 30; the correctness of Algorithm 31 follows similarly. By construction, cf. Algorithm 12, the 3-tuple $(A_1, A_2, \mathbf{Y}_i)$ in line 4 of Algorithm 30 holds and ElGamal encryption of $\mathbf{AA_{Di}}[i]^{-1} \cdot P_1$ under public key Y_i where P_1 is the OAEP embedding of the user identity. By the third part of Proposition 4.1, the 3-tuple E_2 in line 8 holds an ElGamal encryption of

$$(\mathbf{AA_{Di}}[i] \cdot \mathbf{AA_{Di}}[j]^{-1}) \cdot \mathbf{AA_{Di}}[i]^{-1} \cdot P_1 = \mathbf{AA_{Di}}[j]^{-1} \cdot P_1$$

under public key Y_i. By the second part of Proposition 4.1, the 3-tuple E_3 in line 9 holds an ElGamal encryption of $\mathbf{AA_{Di}}[j]^{-1} \cdot P_1$ under public key $\Delta y_{i,j} \cdot Y_i$. By Equation (26) this public key is equal to Y_j. It follows that Algorithm 30 return a polymorphic identity corresponding to Y_j matching the Keygroup #4 version of $\mathbf{ID_{Pi}}$.

Key deactivation time
When the keys in Keygroup #4 are renewed, then all keys with the exception of (the previously issued) keys of type $\mathbf{AA_M}, \mathbf{AA_{Di}}$ can be deactivated and in fact destroyed at all parties. The key of type $\mathbf{AA_M}$ needs to stay active for handling requests of the supervisor for decrypting audit blocks. Also, the key of type $\mathbf{AA_{Di}}$ needs to stay active as long as there are actively used polymorphic identities or pseudonyms based on it. As introduced above, we let C denote the maximum validity of a polymorphic form. Then after R+C years the key $\mathbf{AA_{Di}}$ can be deactivated. We note that R+C years after activation of the keys $\mathbf{y}, \mathbf{Y}, \mathbf{z}, \mathbf{Z}$ there are no longer active polymorphic forms based on these keys.

Suggestions on roll-over and deactivation time periods
Keys of type $\mathbf{y}, \mathbf{Y}, \mathbf{z}, \mathbf{Z}$ resemble the private signing key of a PKI root Certification Authority and the electronic passport Country Signing Certification Authority (CSCA), cf. [26]. The maximal validity period R+C then corresponds with the life validity period of a root Certification Authority. Dutch government requirements, cf. [45], stipulate a maximal validity period of 15 years for such keys. We suggest a validity (deactivation time) period (R+C) of 18 years that is somewhat longer. Given that $C = 10$, this implies a roll-over period R of 8 years.

10.5 Roll-over for Keygroup #5

The keys in Keygroup #5 are generated and maintained by the KMA and correspond to the delivered user pseudonyms at the service providers. By only changing the $\mathbf{PC_M}$ key (Pseudonym Closing Master key) service providers are issued different closing keys of type $\mathbf{PC_{Di}}$ which will lead to new pseudonyms. The conversion between the old and new pseudonyms is addressed in Algorithm 27 in Section 9. If a service provider chooses not implement the new closing key $\mathbf{PC_{Di}}$ the delivered pseudonyms stay the same. However, when the key $\mathbf{PS_M}$ is changed the service provider does have such a choice and the pseudonyms delivered by authentication providers will change. Indeed, by a $\mathbf{PS_M}$ change the ephemeral Pseudonym Shuffle Derived key $\mathbf{PS_{Di,R}}$ applied by the authentication provider will change and thus the delivered pseudonym. This directly follows from Formula (19).

After a period of P years following the previous activation, the KMA generates new keys of type $\mathbf{PS_M}, \mathbf{PC_M}$ and announces their existence in the scheme metadata including their key versions and activation time. We let $\mathbf{PS_M}', \mathbf{PC_M}'$ denote the new keys. The KMA makes the new key $\mathbf{PS_M}'$ available in the HSMs of the authentication providers coupled with their key fields including key versions and activation time. Based on this key the authentication providers can then form the new ephemeral Pseudonym Shuffle Derived key $\mathbf{PS_{Di,R}}'$. When the activation time is reached both the KMA and authentication providers start using the new keys. The KMA uses the new $\mathbf{PC_M}'$ key to deliver new closing keys to service providers.

We now indicate how a service provider can conveniently migrate to new keys in Keygroup #5 by using Algorithm 27 from Section 9. During a migration period, the service provider can, as part of Algorithms 19, 20, request the authenticator provider for pseudonyms based on the previous (old) or next (new) key $\mathbf{PS_M}'$ by referring to the key version of the $\mathbf{PC_M}'$ key. In the first situation we require that the service provider still uses a closing key based on the previous key $\mathbf{PC_M}$. When a service provider indicates he wants to migrate, it provided two keys by KMA: a new closing key which we denote with $\mathbf{PC_{Di}}'$ and a conversion key Γ_M. Prior to using the new closing key $\mathbf{PC_{Di}}'$ the service provider needs to convert it pseudonymous registration according to Algorithm 27 using conversion key Γ_M.

Although the actual usage of the key Γ_M by the service provider is the same as with a closing key migration, the mathematical form of the Γ_M is different:

$$\Gamma_M = \frac{\mathbf{PC_{Di}}' \cdot \mathbf{PS_{Di,R}}'}{\mathbf{PC_{Di}} \cdot \mathbf{PS_{Di,R}}}. \tag{27}$$

It easily follows from Formula (19) that applying Algorithm 27 with key Γ_M on an old pseudonym P leads to a correctly formed pseudonym P'. Indeed, this

algorithm will lead to $P' = \Gamma_M \cdot P$. We now have:

$$\begin{aligned}\Gamma_M \cdot P &= \frac{\mathbf{PC_{Di}}' \cdot \mathbf{PS_{Di,R}}'}{\mathbf{PC_{Di}} \cdot \mathbf{PS_{Di,R}}} \cdot \mathbf{PC_{Di}} \cdot \mathbf{PS_{Di,R}} \cdot \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \boldsymbol{\mathcal{W}}(\mathbf{IW_M}, Id) \\ &= \mathbf{PC_{Di}}' \cdot \mathbf{PS_{Di,R}}' \cdot \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot \boldsymbol{\mathcal{W}}(\mathbf{IW_M}, Id) \\ &= P'.\end{aligned}$$

Here the first equality follows from Formula (27) and Formula (19) with the old keys. The second equality is trivial. The third equality is another application of Formulae (19) with the new keys.

Key deactivation time
During pseudonym migration one wants to avoid a "big bang" in which all service provider need to migrate to the new pseudonyms in a short time. We therefore distinguish a pseudonym migration period in years, denoted by M, in which service providers have to migrate to these new keys leading to different pseudonyms. It is best if service providers pseudonym migration is centrally planned. It is best to start with only a few, most proficient service providers. Then P+M years after activation of the keys $\mathbf{PS_M}, \mathbf{PC_M}$ they can be deactivated.

Suggestions on roll-over and deactivation time periods
To reduce the re-identification risk it is good practice for pseudonymous registrations to periodically change the pseudonyms used. We are not aware of common practices or standards on mandatory pseudonym changes in personal data registrations. Pseudonym change is tackled in the draft Dutch standard NEN 7524 entitled "Health informatics - Pseudonymization service" of February 2019. See www.nen.nl. As an example this draft standard suggests that pseudonyms are changed every five years. The only situation we know of where mandatory pseudonym change occurs, is the German eID scheme. In this context, the user pseudonym is technically linked to the eID card: when a user gets a new eID card, the user gets new pseudonyms at service providers. As the German eID card is valid for 10 years, the pseudonyms change every 10 years. Based on the draft standard NEN 7524 and the German eID practice we suggest a pseudonym change every 10 years in the polymorphic eID scheme. That is, we suggest that $P = 10$. We further suggest a migration period of $M = 2$ year. That is, we suggest a deactivation period of the keys $\mathbf{PS_M}', \mathbf{PC_M}'$ of 12 years.

10.6 Roll-over for Keygroup #6

The keys $\mathbf{IW_M}, \mathbf{IM_M}$ in Keygroup #6 are generated and maintained by the KMA and correspond to the base pseudonyms generated by BSN-L which are later transformed by the authentication providers and service providers to the final pseudonyms. Compare Formulae (19). A change of either keys $\mathbf{IW_M}, \mathbf{IM_M}$ leads to fundamentally new pseudonyms which do not allow for conversion algorithms like Algorithm 27 used in Section 10.5. Due to the fundamental role of the $\mathbf{IW_M}, \mathbf{IM_M}$ keys we choose to let their roll-over to be of roll-over of

all keys in the eID scheme. Actually, in this process one could also change the elliptic curve, e.g. from the Brainpool320r1 curve to the Brainpool384r1 or Brainpool512r1 curve, or even change to another group $\mathbb{G} = (\langle G \rangle, +)$.

10.7 Roll-over to all new keys

Work in progress.

11 References

1. Article 29 Data Protection Working Party, Opinion 05/2014 on Anonymisation Techniques, 10 April 2014.
2. M. Bellare, Mihir, Rogaway, Phillip, Random Oracles are Practical: A Paradigm for Designing Efficient Protocols, ACM Conference on Computer and Communications Security, 1995, pp.62–73.
3. Bellare, M. and P. Rogaway, Optimal Asymmetric Encryption - How to Encrypt with RSA, Eurocrypt, Lecture Notes in Computer Science, Volume 950, pp. 92-111, November 1995.
4. Bundesamt für Sicherheit in der Informationstechnik (BSI), TR-03110-1, Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 1– eMRTDs with BAC/PACEv2 and EACv1, version 2.10, 20 March 2012
5. Bundesamt für Sicherheit in der Informationstechnik (BSI), TR-03110-2, Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 2 – Protocols for electronic Identification, Authentication and trust Services (eIDAS), Bundesamt für Sicherheit in der Informationstechnik (BSI), version 2.21, 21 December 2016
6. Bundesamt für Sicherheit in der Informationstechnik (BSI), TR-03110-3, Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 3 – Common Specifications, Bundesamt für Sicherheit in der Informationstechnik (BSI), version 2.21, 21 December 2016
7. Bundesamt für Sicherheit in der Informationstechnik (BSI), TR-03110-4, Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 4 – Applications and Document Profiles, Bundesamt für Sicherheit in der Informationstechnik (BSI), version 2.21, 21 December 2016
8. Bundesamt für Sicherheit in der Informationstechnik (BSI), TR-03124-1, eID-Client –Part 1: Specifications, version 1.3, 12 June 2017.
9. Bundesamt für Sicherheit in der Informationstechnik (BSI), TR-03130-1, eID-Server –Part 1: Functional Specification, version 2.0.2, 16 November 2016.
10. www.personalausweisportal.de/DE/Wirtschaft/Technik/eID-Service/eID-Service_node.html
11. Bundesamt für Sicherheit in der Informationstechnik, Elliptic Curve Cryptography, TR-03111, version 2.10, 2018-06-01, 2018.
12. Bundesamt für Sicherheit in der Informationstechnik (BSI), Group Signatures: Authentication with Privacy, 2012
13. Cooper, David; Santesson, Stefan; Farrell, Stephen; Boeyen, Sharon; Housley, Russell and Polk, Tim. Internet X.509 public key infrastructure - certificate and certificate revocation list (CRL) profile, RFC 5280, 2008
14. Department of Health and Human Services, Standards for Privacy of Individually Identifiable Health Information; Final Rule, Federal Register / Vol. 67, No. 157 / Wednesday, August 14, 2002 / Rules and Regulations.
15. Dutch Data Protection Authority, Pseudonimisering risicoverevening, 6 March 2007. Reference: z2006-1382.
16. ECC Brainpool, ECC Brainpool standard curves and curve generation, October 2005. See http://www.ecc-brainpool.org.
17. European Parliament and the Council of the European Union, on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), regulation 2016/679.

18. European Parliament and the Council of the European Union, Electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC, regulation 910/2014 2014.
19. European Commission (2015), eIDAS implementing regulation 2015/1501.
20. Official Journal of the European Union, Electronic identification schemes notified pursuant to Article 9(1) of Regulation (EU) No 910/2014 of the European Parliament and of the Council on electronic identification and trust services for electronic transactions in the internal market, 2017/C 319/03, volume 60, 26 September 2017.
21. European Telecommunications Standards Institute (ETSI), Electronic Signatures and Infrastructures (ESI); Cryptographic Suites, TS 119312, V1.2.1, 2017-05.
22. T. ElGamal, A Public Key Cryptosystem and a Signature scheme Based on Discrete Logarithms, IEEE Transactions on Information Theory 31(4), 1985, pp. 469-472.
23. A. Fiat, A. Shamir, How To Prove Yourself: Practical Solutions to Identification and Signature Problems, Crypto, Lecture Notes in Computer Science, Volume 263, 1986, p. 186-194.
24. S. Fründt, Nur fünf Prozent nutzen den E-Perso im Internet, Welt, 7 June 2015. See https://www.welt.de.
25. D. Hankerson, A. Menezes and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, 2004.
26. International Civil Aviation Organization (ICAO), Doc 9303, Machine Readable Travel Document, Seventh Edition, 2015.
27. IETF, The application/json Media Type for JavaScript Object Notation (JSON), July 2006.
28. IETF, PKCS #1: RSA Cryptography Specifications, version 2.2, November 2016.
29. ISO/IEC 7816-4, Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange, third edition, 2013-04-15.
30. ISO/IEC14888-3, Information technology - Security techniques - Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms, 2016.
31. J. Katz, Y. Lindell, Introduction to Modern Cryptography, CRC PRESS, 2008.
32. K. Kurosawa, Multi-Recipient Public-Key Encryption with Shortened Ciphertext, PKC 2002, Lecture Notes in Computer Science, Volume 2274 pp 48-63, 2002.
33. Mazars, Privacy Impact Assessment, Introductieplateau eID Stelsel NL (versie 0.8), version 1.0, 31 July 2015.
34. NIST, Advanced Encryption Standard (AES), FIPS PUB 197, November 2001.
35. NIST, Secure Hash Standard (SHS), FIPS PUB 180-4, August 2015.
36. National Institute for Standards and Technology, Digital signature standard, FIPS PUB 186-4. July 2013 .
37. NIST, The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198-1, July 2008.
38. NIST, Recommendation for Key Management Part 1: General, January 2016.
39. NIST, Digital Authentication Guideline, Federation and Assertions. NIST Special Publication 800-63C, June 2017.
40. NIST, Recommendation for Key Derivation Using Pseudorandom Functions, SP 800-108, October 2009.
41. OASIS, (2005), Security Assertion Markup Language (SAML).
42. OASIS, PKCS #11 Cryptographic Token Interface Base Specification, version 2.4, 14 April 2015.
43. PKIOverheid, Programma van Eisen deel 3g: Certificate Policy Authenticiteit en Vertrouwelijkheidcertificaten – Domein Private Services, version 4.7, 8 February 2019.

44. PKIOverheid, Certification Practice Statement (CPS) Policy Authority PKIoverheid for G2 and G3 CA certificates to be issued by the Policy Authority of the PKI for the Dutch government, version 1.4, December 2018.
45. PKIOverheid, Certification Practice Statement (CPS) Policy Authority PKIoverheid for Private Root CA certificates to be issued by the Policy Authority of the PKI for the Dutch government, version 1.7, December 2018.
46. C. P. Schnorr, Efficient signature generation for smart cards. Journal of Cryptology, 4(3):239-252, 1991.
47. L. Sweeney. k-Anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-based Systems, 10:557– 570, 2002.
48. E. Verheul, Polymorphic Pseudonymization, version 0.91, 07 July 2014.
49. E. Verheul, B. Jacobs, Polymorphic encryption and pseudonymisation in identity management and medical research, Nieuw Archief voor Wiskunde (NAW), 5/18 nr. 3, September 2017
50. E.R. Verheul, H.C.A. van Tilborg, Binding ElGamal: A Fraud-Detectable Alternative to Key-Escrow Proposals, EUROCRYPT 1997 , Lecture Notes in Computer Science, Volume 1233, pp. 119-133, 1997.
51. E.R. Verheul, Evidence that XTR is more secure than supersingular elliptic curve cryptosystems, Journal of Cryptology (JOC) 17(4), pp. 277-296, 2004.
52. World Wide Web Consortium (W3C), Extensible Markup Language (XML). See http://www.w3.org/XML/.

A Glossary of terms and abbreviations

Term	Explanation
@, [.]	Used as (D)EP@X or (D)EP@[X,Y] indicating (D)EP at X and (D)EP at X and Y.
$\|\|$	Regular concatenation of byte arrays or strings.
$\|\|\|$	Injective concatenation or byte arrays or strings, i.e. different inputs always result in different concatenation results. In practice this is usually catered for by using a special separating identifiers that are not allowed to occur in the input.
AB	Audit Block part of polymorphic and encrypted forms containing encrypted data for the Supervisor.
AP, AP_{ID}	Authentication Provider, a (private) party that authenticates a user for Service Providers resulting in either a global identity (typically the social security number) or a pseudonym. For this an AP *transforms* polymorphic forms (PIs, PPs or PIPs) into encrypted forms (EIs or EPs). As part of its role AP issues authenticators to users and links them to polymorphic forms. Each AP is uniquely with an identifier AP_{ID}.
Audit Block	A structure consisting of 16 byte holding an AES encryption of the concatenation of a 4 byte HSM identifier (HSM_{ID}), a 4 byte time indicator T (e.g. holding seconds since 1 January 1970 UTC) and a 8 byte serial number (SN).
Authenticator	Technical means of authentication provided by the AP to the user, allowing him to authenticate to the AP. Examples are a smart card, a hardware token and an authentication APP.
Base Pseudonym	The keyed derivative of the user identity (typically BSN) placed inside a PP.
BSN	Dutch social security number (burgerservicenummer).
BSN-L	BSN Linking service, generates polymorphic forms (PIs or PPs) for authentication providers as part of user activation. As part of this, BSN-L also generates DEPs for the user inspection register.
Cryptogram Identifier (CrId)	See Section 4.9.
DEP, DEP_{ID}	Direct Encrypted Pseudonym, cryptogram generated by BSN-L sent to the user inspection register from which the latter can decrypt the local pseudonym of a user. A DEP cryptogram is associated with cryptogram identifier DEP_{ID}.
DEI, DEI_{ID}	Direct Encrypted Identity, cryptogram generated by BSN-L that can optionally be sent to an authentication provider as part of activation. It allows the authentication provider to periodically repeat the activation process without the user identity (BSN).
EI, EI_{ID}	Encrypted Identity, cryptogram sent by AP to SP from which he can decrypt the global identity of a user. An EI cryptogram type is associated with object identifier EI_{ID}.
EP, EP_{ID}	Encrypted Pseudonym, cryptogram sent by AP to SP from which he can decrypt the local pseudonym of a user. An EP cryptogram type is associated with object identifier EP_{ID}.

Term	Explanation
ERT	Election Result Table
$\mathbb{G}, \mathbb{G}^*$	$\mathbb{G} = (\langle G \rangle, +)$ is an additive group of order q generated by a generator element G. We assume $\mathbb{G}$ is an elliptic curve group. $\mathbb{G}^*$ denotes the non-zero elements of $\mathbb{G}$.
h	OAEP hash length in bytes i.e. 10 in version 1 of the polymorphic eID scheme.
HSM, HSM_{ID}	Hardware Security Module with its unique identifier consisting of 4 byte in scheme version 1.
Id	Identity, unique identifier of a user (citizen), typically its social security number (BSN). The scheme supports that users have multiple identities as long as different citizens can never have the same identity.
k	Size in bytes of the prime number p over which the elliptic curve group size is defined, i.e. 40, in version 1 of the polymorphic eID scheme.
KDF	Key Derivation Function, a function that deterministically generate a secret key based on a master key and a derivation string.
Key Identifier (KId)	See Section 4.9.
KMA	Key Management Authority, party that generates and provides eID participants with cryptographic keys.
KV	Key specific version string allowing key versioning in the scheme. Every key has its own key version string and which can be empty. See the end of Section 4.6.
OAEP	Optimal Asymmetric Encryption Padding
l	Size in bits of q, i.e. 320 in version 1 of the polymorphic eID scheme.
m	Maximal length of identifiers, e.g. BSN, in bytes (relevant for OAEP). This is 18 in version 1 of the polymorphic eID scheme.
p	Prime number over which the elliptic curve group $\mathbb{G}$ is defined.
Participant	All parties except the KMA.
PI, PI_{ID}	Polymorphic Identity, encryption of a global identity (typically the social security number) that can be transformed to an EI. A PI cryptogram type is associated with object identifier PI_{ID}.
PIP, PIP_{ID}	Polymorphic Identity and Pseudonym, data efficient combination of PI and PP. A PIP cryptogram type is associated with object identifier PIP_{ID}.
PP, PP_{ID}	Polymorphic Pseudonym, global encryption of a base pseudonym that can be transformed to an EP. A PP cryptogram type is associated with object identifier PP_{ID}.
q	Prime order of group $\mathbb{G}$.
R	Role, a byte array or string that represents a user role at a service provider, e.g. in case or representation. The role will be used in the pseudonym forming for the service provider. In most applications R will be empty.
RVA	Register Voters Abroad
Scheme Key	A cryptographic key whose change implies migration activities for (almost) all parties.
Scheme Metadata	Data allowing parties to obtain security relevant data of parties that participate in the eID scheme including cryptographic data.

Term	Explanation
SP, SP_{ID}	Service Provider, a (private) party that provides electronic services to (authenticated) users. Each SP is uniquely with an identifier SP_{ID}.
SM	Secure Messaging
SN	Sequence Number, in eID scheme version 1 this is an eight byte number.
SS	Status Service, used within the PCA card for both black as white listing.
T	Bye array holding time (4 byte), e.g. holding seconds since 1 January 1970 UTC.
$\tilde{T}$	Byte array holding reduced time (3 byte), e.g. BCD encoding of year and month formed as YYYYMM.
Unclosed pseudonym	Pseudonym as delivered by authentication provider. Compare Formula (19).
TLP	Transaction Log Provider, a service provider to which an AP sends transactions related to performed authentication. This could a separated part of the AP or could be placed at a different party.
UIS	User Inspection Service, service provider where citizens can assess the APs where they have activated an authenticator. The UIS is directly informed by BSN-L on activations using DEPs.

B Representing identity strings as byte arrays

We specify two representations as a byte array of an identity string Id . In both representations the type T of the identity string is also included and we assume that this takes the form of a byte value. The type of an identity string also specifies its canonical byte array representation. The value of the Dutch social security number (called BSN) is 0x42 which is the ASCII value of the character B. The value for the eIDAS Uniqueness Identifier of [18] is 0x55 which is the ASCII value of the character U. We let l_{Id} denote the length of the byte array representation of Id using its canonical representation, e.g. ASCII values when Id consist of printable ASCII characters. We assume that l_{Id} fits a byte, i.e. is less then 256. Below we specify representation $D(.)$ used in Section 4.7 and representation $I(.)$ used in Section 4.8. Representation $I(.)$ latter takes as input Id and T and returns a byte array of size of Id plus two. Representation $D(.)$ takes as input Id, T and an integer m and returns a byte array of size of m. The integer m should not be smaller than the size of Id plus three.

Algorithm 32 $E(Id, T, m)$
Encoding an identity string Id of type T as a byte array of size m.

1: `Validate` Id `is of type` T`, otherwise return Error` `// input validation`
2: `If` $m < l_{Id} + 3$ `return Error`
3: `Let` B `be a byte array of size` m
4: `Set` $B[0] = 0x01$ `// versioning`
5: `Set` $B[1] = T$
6: `Set` $B[2] = l_{Id}$
7: `Copy the canonical byte representation of` Id `to` $B[3], \ldots, B[3 + l_{Id} - 1]$.
8: `Set remaining bytes` $B[3 + l_{Id}] \ldots, B[m - 1]$ `to zero`
9: `Return` B

As an illustration, the $E(Id, 42, 18)$ value of the Dutch social security number "999990019" is the byte array

$$\{\, 0x01\ , 0x42, 0x09, 0x39, 0x39, 0x39, 0x39, 0x39, 0x30,$$
$$0x30\ , 0x31, 0x39, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00\}.$$

Algorithm 33 $D(B, m)$
Decoding a byte array of size m to an identity string Id and type T.

1: `If size of` B `is not equal to` m `return Error`
2: `If` $B[0] \neq 0x01$ `return Error` `// versioning`
3: `Set` $B[1] = T$
4: `Set` $B[2] = l$
5: `If any` $B[3 + l_{Id}] \ldots, B[m - 1]$ `is non-zero return Error`
6: `Interpret` $B[3], \ldots, B[3+l-1]$ `as identity string` Id`, on failure return Error`
7: `Return` Id `and` T

The check in Line 5 of Algorithm 33 is an extension the OAEP decoding validations.

Algorithm 34 $I(Id, T)$
Mapping an identity string Id of type T to a byte array of size $l_{Id} + 2$.

1: `If` $m < l_{Id} + 3$ `return Error`
2: `Let` B `be a byte array of size` $l_{Id} + 2$
3: `Set` $B[0] = 0x01$ `// versioning`
4: `Set` $B[1] = T$
5: `Copy the canonical byte representation of` Id `to` $B[2], \ldots, B[2 + l_{Id} - 1]$.
6: `Return` B

As an illustration, the $I(Id, 42)$ value of the Dutch social security number "999990019" is the byte array

$$\{0x01, 0x42, 0x39, 0x39, 0x39, 0x39, 0x39, 0x30, 0x30, 0x31, 0x39\}.$$

C Cryptographic keys in use

C.1 Key types and algorithms

Table 6 below lists all cryptographic keys used in the polymorphic eID scheme cryptographic specification, i.e. in Section 5:

- Column two gives the Key ID for the key, i.e. an abbreviated name for the key. Here public keys are coloured **green** whereas secret or private keys are coloured **red**. For all keys except the signing keys of BSN-L, this name takes the form $XY_{Z[i]}$ for which a mnemonic is provided in Table 5 below.
- Column three contains the full name for the key related to the mnemonics.
- Column four indicates the key type and the algorithm it supports. This is further elaborated on below.
- Column five contains a short description of the key function.
- Column six indicates the party that generates the key and column seven the parties using it (other than the generator).
- Column eight indicates if the key is a *Global Key*. This is a key whose change implies migration activities for (almost) all parties. The number of eight global keys is eight counting a public/private pair as one.
- The rightmost part of the table indicates a grouping of the keys and its primary user. This follows the life cycle in time of the process leading to an Identity/Pseudonym at a service providers.

As indicated in the third column of Table 6, several keys within the scheme are derived from a master key. Compare Section 4.6. This is further specified in Table 7 of Appendix C.2.

X	Y	Z	[i]
A = Authentication provider	A = Adherence	D = Derived key	i indicates that
I = Identity	D = Decryption	K = Key (no diversification)	several (different)
P = Pseudonym	E = Encryption	M = Master key	instances exist
D = Direct	M = Mapping	P = Public key	
S = Supervisor	P = Private	a = Activation	
	R = Receiving	t = Transformation	
	S = Shuffle		
	T = Transmission		
	W = Wrapping		

Table 5. Mnemonic for key ID of form $XY_{Z[i]}$

#	KId	Name	Type	Function	Generator	Users	Global Key	Derived Key
1.	y (or IP_M)	Identity Private Master key	EC Private Key (ElGamal)	Generation of SP EI decryption keys	KMA		Yes	No
2.	Y (or IP_P)	Identity Private Public key	EC Public Key (ElGamal)	PI generation	KMA	BSN-L	Yes	No
3.	z (or PP_M)	Pseudonym Private Master key	EC Private Key (ElGamal)	Generation of SP EP decryption keys	KMA		Yes	No
4.	Z (or PP_P)	Pseudonym Private Public key	EC Public Key (ElGamal)	PP generation	KMA	BSN-L	Yes	No
5.	PC_M	Pseudonym Closing Master key	HMAC key (Master key)	Generation of SP closing keys	KMA		Yes	No
6.	DC_M	Direct Communication Master Key	HMAC key (Master key)	DEP deployment (special SPs only)	KMA		Yes	No
7.	IW_M	Identity Wrapping Master key	HMAC key (Master key)	PP generation (map into curve)	KMA	BSN-L	Yes	No
8.	IM_M	Identity Mapping Master key	HMAC key (Master key)	PP generation (map inside curve)	KMA	BSN-L	Yes	No
9.	AA_M	Authentication provider Adherence Master key	HMAC key (Master key)	PP/PI generation (make AP specific)	KMA	BSN-L	Yes	No
10.	AA_{Di}	Authentication provider Adherence Derived key	HMAC key (Master key)	Polymorphic transform	KMA	BSN-L APs	No	Yes
11.	$DT_{Di,R}$	Direct Transmission Derived key	EC Private key (Diffie-Hellman) EC Public Key (ElGamal)	DEP generation by BSN-L	KMA	BSN-L	No	Yes
12.	u	PI/PP Signing key	EC Private Key (ECDSA)	Signing of PI, PP DEP and DEI	BSN-L	BSN-L	No	No
13.	U	PI/PP Verification key	EC Public Key (ECDSA)	Verifying of PI, PP DEP and DEI	BSN-L	APs	No	No
14.	IE_M	Identity Encryption Master key	HMAC key (Master key)	PI to EI transform	KMA	APs	Yes	No
15.	IE_{Di}	Identity Encryption Derived key(ephemeral)	EC Private key (Diffie-Hellman)	PI to EI transform	KMA	APs	Yes	Yes
16.	PE_M	Pseudonym Encryption Master key	HMAC key (Master key)	PP to EP transform	KMA	APs	Yes	No
17.	PE_{Di}	Pseudonym Encryption Derived key(ephemeral)	EC Private key (Diffie-Hellman)	PP to EP transform	KMA	APs	Yes	Yes
18.	PS_M	Pseudonym Shuffle Master key	HMAC key (Master key)	PP to EP transform	KMA	APs	Yes	No
19.	$PS_{Di,R}$	Pseudonym Shuffle Derived key(ephemeral)	EC Private key (Diffie-Hellman)	PP to EP transform	KMA	APs	Yes	Yes
20.	ID_{Di}	Identity Decryption Derived key	EC Private key (ElGamal)	EI to I decryption	KMA	SPs	No	Yes
21.	ID_{Pi}	Identity Decryption Public key	EC Public key (ElGamal)	EI validation (EC-SCHNORR)	KMA	SPs	No	Yes
22.	PD_{Di}	Pseudonym Decryption Derived key	EC Private key (ElGamal)	EP to P decryption	KMA	SPs	No	Yes
23.	PD_{Pi}	Pseudonym Decryption Public key	EC Public key (ElGamal)	EP validation (EC-SCHNORR)	KMA	SPs	No	Yes
24.	$DR_{Di,R}$	Direct Receiving Derived key	EC Private key (Diffie-Hellman) EC Private Key (ElGamal)	DEP to EP (special SPs only)	KMA	SPs	No	Yes
25.	PC_{Di}	Pseudonym Closing Derived key	EC Private key (Diffie-Hellman)	EP to P decryption	KMA	SPs	No	Yes
26.	SED_a	Supervisor Encryption Derived key	AES key	Auditelement decryption	KMA	BSN-L Supervisor	No	Yes
27.	SED_t	Supervisor Encryption Derived key	AES key	Auditelement decryption	KMA	APs Supervisor	No	Yes

Table 6. eID (Scheme) Keys

Table 6 indicates the following key types and algorithms:

EC public / private keys Private keys are of type $x \in_R \mathbb{F}_q^*$ and public keys are of type $x \cdot G$ in the context of Section 4.1. In version 1 of the polymorphic scheme private keys are 320 random integers and public keys points on the Brainpool320r1 curve. These types of keys are used in ElGamal encryption (Section 4.2) and in digital signatures (Section 4.3).

HMAC keys These are random binary strings of size $\|q\|$, i.e. the length of the group order in bits.In version 1 of the polymorphic scheme private keys this length is 320. That is, here HMAC keys are byte arrays of size 40 bytes. These types of keys are used in the key derivation functions we use (Section 4.6).

AES keys Random binary strings of size 256 bits, see Section 4.5.

C.2 Derived keys

Table 7 below specifies how derived keys in the scheme are generated from master keys and derivation data. As indicated in this table we use derivation functions $\mathcal{K}_1(.)$ and $\mathcal{K}_3(.)$ from Section 4.6. The derivation input data takes the form of strings which we implicitly interpret as byte arrays using their cannonical representation (without trailing zero byte). See Section 4.6. As indicated, derivation input strings are formed through a particular concatenation of input strings. All such input strings are assumed to be non-empty. We let KV represents a key version integer allowing key versioning in the scheme, see Section 4.9. We note that key versions are key specific: every key has its own key version. Key versions in Table 7 are represented as strings by their unsigned decimal string representation, e.g. the integer 1 is represented by the string "1". The plain use of KV in Table 7 refers to the key version of the key constructed which is chosen by the party generating the key. As an illustration, the KV in the first row of Table 7 relates to the key version of the key $\mathbf{AA_{Di}}$ being constructed. The generation of $\mathbf{AA_{Di}}$ is also an example of a key derivation that only uses the key version of the key generated. This is commonly the case, exceptions are derived keys such as $\mathbf{ID_{Pi}}, \mathbf{ID_{Di}}$ that also depend of key versions of key types $\mathbf{y}, \mathbf{Y}$ or $\mathbf{z}, \mathbf{Z}$. This approach allows for securely reusing the same $\mathbf{IE_M}, \mathbf{PE_M}$ keys with different $\mathbf{y}, \mathbf{z}$ which is essential in key roll-over, see Section 10. Without including key versions of the keys $\mathbf{y}, \mathbf{z}$, keys like $\mathbf{ID_{Pi}}, \mathbf{PD_{Pi}}$ would leak secret information on the quotients of successive $\mathbf{y}, \mathbf{z}$ keys.

We require that concatenation of the input strings is injective, i.e. that the concatenation of different input strings always results in a different output string. This is indicated by using the $\|\|$ symbol instead of the regular concatenation symbol $||$. We simply arrange for this by using special separating non-alphanumeric ASCII symbols that are not allowed to occur in the input strings. This is indicated in the last column of Table 7. As an illustration, for key version 1 and identifier $\mathrm{AP_{ID}}$ equal to "DigiD", the input string used in the derivation of $\mathbf{AA_{Di}}$ is "DigiD@1". Its byte array representation is

$$\{0x44, 0x69, 0x67, 0x69, 0x44, 0x40, 0x31\}.$$

Special attention needs to be given to the derivation of the keys of type $\mathbf{SED_a}$ and $\mathbf{SED_t}$, as these use master keys $\mathbf{AA_M}$ and $\mathbf{PE_M}$ also used in the derivation of $\mathbf{AA_{Di}}$ and $\mathbf{PD_{Di}}, \mathbf{PD_{Pi}}$. To this end we use different separating symbols in the derivation of $\mathbf{SED_a}$, respectively $\mathbf{SED_t}$, and in the derivation of $\mathbf{AA_{Di}}$, respectively $\mathbf{PD_{Di}}, \mathbf{PD_{Pi}}$. As an illustration, for key version 2, identifier $\mathrm{AP_{ID}}$ equal to "DigiD" and identifier $\mathrm{Sup_{ID}}$ equal to "AT" the string input used in the derivation of $\mathbf{SED_t}$ is equal to "AT#DigiD#2". Its byte array representation is

$$\{0x41, 0x54, 0x23, 0x44, 0x69, 0x67, 0x69, 0x44, 0x23, 0x32\}.$$

KId	Derived Key Name	Relevant key data (simplified)	Derivation formula	Separator
10.	$\mathbf{AA_{Di}}$	KVS: $\mathbf{AA_M}$ Recipient: $\mathrm{AP_{ID}}$	$\mathcal{K}_1(\mathbf{AA_M}, \text{Recipient} \,\|\|\| \text{KV})^{-1}$	@ (0x40)
11.	$\mathbf{DT_{Di,R}}$	KVS: $\mathbf{Z}, \mathbf{PE_M}, \mathbf{PS_M}, \mathbf{DC_M}$ Creator: BSN-L_{ID} Recipient: $\mathrm{SP_{ID}}$ Role: N/A	$\text{Keyd}[0] = \mathcal{K}_1(\mathbf{DC_M}, \text{Creator} \,\|\|\| \text{Recipient} \,\|\|\| \text{KV})$ $\text{Keyd}[1] = \mathcal{K}_1(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{Z}.\text{KV}) \cdot \mathbf{Z}$ $= \mathbf{PD_{Pi}}$	@ (0x40)
		KVS: $\mathbf{Z}, \mathbf{PE_M}, \mathbf{PS_M}, \mathbf{DC_M}$ Creator: BSN-L_{ID} Recipient: $\mathrm{SP_{ID}}$ Role: R	$\text{Keyd}[0] = \mathcal{K}_1(\mathbf{DC_M}, \text{Creator} \,\|\|\| \text{Role} \,\|\|\| \text{Recipient} \,\|\|\| \text{KV})$ $\text{Keyd}[1] = \mathcal{K}_1(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{Z}.\text{KV}) \cdot \mathbf{Z}$ $= \mathbf{PD_{Pi}}$	@ (0x40)
15.	$\mathbf{IE_{Di}}$ (eph)	KVS: $\mathbf{Y}, \mathbf{IE_M}, \mathbf{ID_{Pi}}$	$\mathcal{K}_1(\mathbf{IE_M}, \mathbf{ID_{Pi}}.\text{Recipient} \,\|\|\| \mathbf{ID_{Pi}}.\text{KV} \,\|\|\| \mathbf{Y}.\text{KV})$	@ (0x40)
17.	$\mathbf{PE_{Di}}$ (eph)	KVS: $\mathbf{Z}, \mathbf{PE_M}, \mathbf{PD_{Pi}}$	$\mathcal{K}_1(\mathbf{PE_M}, \mathbf{PD_{Pi}}.\text{Recipient} \,\|\|\| \mathbf{PD_{Pi}}.\text{KV} \,\|\|\| \mathbf{Z}.\text{KV})$	@ (0x40)
19.	$\mathbf{PS_{Di,R}}$ (eph)	KVS: $\mathbf{PS_M}, \mathbf{PD_{Pi}}$ Role: N/A	$\mathcal{K}_1(\mathbf{PS_M}, \mathbf{PD_{Pi}}.\text{Recipient})$	-
		KVS: $\mathbf{PS_M}, \mathbf{PD_{Pi}}$ Role: R	$\mathcal{K}_1(\mathbf{PS_M}, \text{Role} \,\|\|\| \mathbf{PD_{Pi}}.\text{Recipient})$	@ (0x40)
20.	$\mathbf{ID_{Di}}$	KVS: $\mathbf{y}, \mathbf{IE_M}$ Recipient: $\mathrm{SP_{ID}}$	$\mathcal{K}_1(\mathbf{IE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{y}.\text{KV}) \cdot \mathbf{y} = \mathbf{IE_{Di}} \cdot \mathbf{y}$	@ (0x40)
21.	$\mathbf{ID_{Pi}}$	KVS: $\mathbf{Y}, \mathbf{IE_M}$ Recipient: $\mathrm{SP_{ID}}$	$\mathcal{K}_1(\mathbf{IE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{Y}.\text{KV}) \cdot \mathbf{Y} = \mathbf{ID_{Di}} \cdot G$	@ (0x40)
22.	$\mathbf{PD_{Di}}$	KVS: $\mathbf{z}, \mathbf{PE_M}$ Recipient: $\mathrm{SP_{ID}}$	$\mathcal{K}_1(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{z}.\text{KV}) \cdot \mathbf{z} = \mathbf{PE_{Di}} \cdot \mathbf{z}$	@ (0x40)
23.	$\mathbf{PD_{Pi}}$	KVS: $\mathbf{Z}, \mathbf{PE_M}$ Recipient: $\mathrm{SP_{ID}}$	$\mathcal{K}_1(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{Z}.\text{KV}) \cdot \mathbf{Z} = \mathbf{PD_{Di}} \cdot G$	@ (0x40)
24.	$\mathbf{DR_{Di,R}}$	KVS: $\mathbf{z}, \mathbf{PE_M}, \mathbf{PS_M}, \mathbf{DC_M}$ Recipient: $\mathrm{SP_{ID}}$ Role: N/A	$\text{Keyd}[0] = \frac{\mathcal{K}_1(\mathbf{PS_M}, \text{Recipient})}{\mathcal{K}_1(\mathbf{DC_M}, \text{Creator} \,\|\|\| \text{Recipient} \,\|\|\| \text{KV})}$ $= \frac{\mathbf{PS_{Di,R}}}{\mathcal{K}_1(\mathbf{DC_M}, \text{Creator} \,\|\|\| \text{Recipient} \,\|\|\| \text{KV})}$ $\text{Keyd}[1] = \mathcal{K}_1(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{z}.\text{KV}) \cdot \mathbf{z}$ $= \mathbf{PD_{Di}}$	@ (0x40)
		KVS: $\mathbf{z}, \mathbf{PE_M}, \mathbf{PS_M}, \mathbf{DC_M}$ Recipient: $\mathrm{SP_{ID}}$ Role: R	$\text{Keyd}[0] = \frac{\mathcal{K}_1(\mathbf{PS_M}, \text{Role} \,\|\|\| \text{Recipient})}{\mathcal{K}_1(\mathbf{DC_M}, \text{Creator} \,\|\|\| \text{Role} \,\|\|\| \text{Recipient} \,\|\|\| \text{KV})}$ $= \frac{\mathbf{PS_{Di,R}}}{\mathcal{K}_1(\mathbf{DC_M}, \text{Creator} \,\|\|\| \text{Role} \,\|\|\| \text{Recipient} \,\|\|\| \text{KV})}$ $\text{Keyd}[1] = \mathcal{K}_1(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{KV} \,\|\|\| \mathbf{z}.\text{KV}) \cdot \mathbf{z}$ $= \mathbf{PD_{Di}}$	@ (0x40)
25.	$\mathbf{PC_{Di}}$	KVS: $\mathbf{PC_M}$ Recipient: $\mathrm{SP_{ID}}$	$\mathcal{K}_1(\mathbf{PC_M}, \mathrm{SP_{ID}} \,\|\|\| \text{KV})$	@ (0x40)
26.	$\mathbf{SED_a}$	KVS: $\mathbf{AA_M}$ Recipient: $\mathrm{Sup_{ID}}$ Auditee: $\mathrm{AP_{ID}}$	$\mathcal{K}_3(\mathbf{AA_M}, \text{Recipient} \,\|\|\| \text{Auditee} \,\|\|\| \text{KV})$	# (0x23)
27.	$\mathbf{SED_t}$	KVS: $\mathbf{PE_M}$ Recipient: $\mathrm{Sup_{ID}}$ Auditee: $\mathrm{AP_{ID}}$	$\mathcal{K}_3(\mathbf{PE_M}, \text{Recipient} \,\|\|\| \text{Auditee} \,\|\|\| \text{KV})$	# (0x23)

Table 7. Key derivation

As can be seen from the above derivation functions, keys of type Pseudonym Decryption Public key ($\mathbf{PD_{Pi}}$) are included as the first part of keys of type Direct Transmission Derived key ($\mathbf{DT_{Di,R}}$). Also keys of type Pseudonym Decryption Private key ($\mathbf{PD_{Di}}$) are included as the first part of keys of type Direct Receiving Derived key ($\mathbf{DR_{Di,R}}$).

D PCA PIP transformation algorithms

We let uPIP_{ID} denote the object identifier associated with an unsigned Polymorphic Identity and Pseudonym also indicating a version number. It is similar with a signed PIP but lacks the audit block and signature. That is a uPIP consists of two elements $(\text{uPIP}_{\text{ID}}, A)$ where A is a sequence of five elliptic curve points. Similarly we let uPP_{ID} denote the object identifier associated with an unsigned Polymorphic Pseudonym also indicating a version number consisting of two elements $(\text{uPP}_{\text{ID}}, A)$ where A is a sequence of three elliptic curve points.

Below are three algorithms specified forming an EI/EP from an unsigned PIP or PP. They are similar to Algorithms 18, 20 and 19, essentially only missing Line 3 where the signature is verified.

Algorithm 35 *GenEI*$(\text{SP}_{\text{ID}}, \text{uPIP})$
AP_{ID} generates Encrypted Identity for SP_{ID} using unsigned $\text{uPIP} = (\text{uPIP}_{\text{ID}}, \text{AP}_{\text{ID}}, A)$.

1: SN=SN+1 // increment sequence number.
2: Validate that $(\text{uPIP}_{\text{ID}}, \text{AP}_{\text{ID}}, A)$ is correctly formed, on failure return Error // input validation
3: Interpret A as 5-tuple $(A_1, A_2, A_3, A_4, A_5) \in \mathbb{G}^5$ // parse PI as EC points
4: Compute $E_1 = \mathcal{RR}(A_1, A_2, A_4)$ // randomize PI
5: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
6: Compute $E_3 = \mathcal{RK}(E_2, \mathcal{K}_1(\mathbf{IE_M}, \text{SP}_{\text{ID}} \,\|\|\, \text{KV}))$ // rekey PI $\rightarrow$ Base EI
7: Form byte array $AB_1 = \text{HSM}_{\text{ID}} \,||\, T \,||\, SN$ // form 16 byte Audit Block
8: $AB_2 = \mathcal{E}_{\text{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
9: Compute $Sig = Sig_{\text{Schn}}(\text{EI}_{\text{ID}} \,||\, \text{SP}_{\text{ID}} \,||\, E_3 \,||\, AB_2, \mathcal{K}_1(\mathbf{IE_M}, \text{SP}_{\text{ID}} \,\|\|\, \text{KV}), \mathbf{Y})$ // sign
10: Return $\text{EI} = (\text{EI}_{\text{ID}}, \text{SP}_{\text{ID}}, E_3, AB_2, Sig)$

Algorithm 36 *GenEP*$(\text{SP}_{\text{ID}}, \text{uPIP})$
AP_{ID} generates Encrypted Pseudonym for SP_{ID} and optional role R using unsigned $\text{uPIP} = (\text{uPIP}_{\text{ID}}, \text{AP}_{\text{ID}}, A)$.

1: SN=SN+1 // increment sequence number
2: Validate that $(\text{PIP}_{\text{ID}}, \text{AP}_{\text{ID}}, A)$ is correctly formed, on failure return Error // input validation
3: Interpret A as 5-tuple $(A_1, A_2, A_3, A_4, A_5) \in \mathbb{G}^5$ // parse PIP as EC points
4: Compute $E_1 = \mathcal{RR}(A_1, A_3, A_5)$ // randomize PP
5: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
6: Compute $E_3 = \mathcal{RK}(E_2, \mathcal{K}_1(\mathbf{PE_M}, \text{SP}_{\text{ID}})$ // rekey PP $\rightarrow$ Base EP.
7: Compute $E_4 = \mathcal{RS}(E_3, \mathcal{K}_1(\mathbf{PS_M}, R \,\|\|\, \text{SP}_{\text{ID}}))$ // reshuffle PP $\rightarrow$ Base EP
8: Form byte array $AB_1 = \text{HSM}_{\text{ID}} \,||\, T \,||\, SN$ // form 16 byte Audit Block
9: $AB_2 = \mathcal{E}_{\text{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
10: Compute $Sig = Sig_{\text{Schn}}(\text{EP}_{\text{ID}} \,||\, \text{SP}_{\text{ID}} \,||\, E_4 \,||\, AB_2, \mathcal{K}_1(\mathbf{PE_M}, \text{SP}_{\text{ID}} \,\|\|\, \text{KV}), \mathbf{Y})$ // sign
11: Return $\text{EP} = (\text{EP}_{\text{ID}}, \text{SP}_{\text{ID}}, E_4, AB_2, Sig)$

Algorithm 37 *GenEP*($\mathrm{SP_{ID}}$, uPP)
$\mathrm{AP_{ID}}$ generates Encrypted Pseudonym for $\mathrm{SP_{ID}}$ and optional role R using unsigned uPP = ($\mathrm{uPP_{ID}}$, $\mathrm{AP_{ID}}$, A).

1: SN=SN+1 // increment sequence number
2: Validate that ($\mathtt{PP_{ID}}, \mathtt{AP_{ID}}, A$) is correctly formed, on failure return Error // input validation
3: Interpret A as 3-tuple $(A_1, A_2, A_3) \in \mathbb{G}^3$ // parse PP as EC points
4: Compute $E_1 = \mathcal{RR}(A_1, A_2, A_3)$ // randomize PI
5: Compute $E_2 = \mathcal{RS}(E_1, \mathbf{AA_{Di}})$ // make AP unspecific
6: Compute $E_3 = \mathcal{RK}(E_2, \mathcal{K}_1(\mathbf{PE_M}, \mathtt{SP_{ID}} \,\|\|\, \mathtt{KV}))$ // rekey PP → Base EP.
7: Compute $E_4 = \mathcal{RS}(E_3, \mathcal{K}_1(\mathbf{PS_M}, R \,\|\|\, \mathtt{SP_{ID}} \,\|\|\, \mathtt{KV}))$ // reshuffle PP → Base EP
8: Form byte array $AB_1 = \mathtt{HSM_{ID}} \,||\, T \,||\, SN$ // form 16 byte Audit Block
9: $AB_2 = \mathcal{E}_{\mathtt{AES}}(AB_1, \mathbf{SED_t})$ // encrypt AB for Supervisor
10: Compute $Sig = Sig_{\mathtt{Schn}}(\mathtt{EP_{ID}} \,||\, \mathtt{SP_{ID}} \,||\, E_4 \,||\, AB_2, \mathcal{K}_1(\mathbf{PE_M}, \mathtt{SP_{ID}} \,\|\|\, \mathtt{KV}), \mathbf{Y})$ // sign
11: Return EP = ($\mathtt{EP_{ID}}, \mathtt{SP_{ID}}, E_4, AB_2, Sig$)

E Verifiable Polymorphic Identity and Pseudonym

E.1 Problem description

As outlined in Section 7.2 the card issuer requests a PIP from the Activation service of BSN-L as part of PCA card production. The PIP are produced by BSN-L using Algorithm 14. The input for this service is the user identity (BSN) and the output is the PIP. We remark that the activation service actually performs certain validations on the BSN not relevant for the present discussion. This PIP, typically only the essential unsigned part of it, is then placed on the PCA card by the card issuer.

In practice card production is often split into two parts: *data preparation* and *card personalization*. The input of data preparation is roughly a list of identities (BSNs) corresponding with the cards that need to be produced. During data preparation all data required for a batch of cards to be produced is gathered; here the PIPs are also requested from BSN-L. This data is then bundled into a data personalization file and then sent to the card personalization process where the data (and often also the card applications) are actually placed onto the cards.

The PIPs produced by Algorithm 14 do not include the BSNs they are based on. This means that during card personalization it cannot be verified that a PIP actually corresponds with the BSN the card is associated with. That is, the personalization process needs to rely on both BSN-L and the data preparation process for this. In theory, BSN-L or the data preparation process could make an error. To mitigate such risk of errors it is common practice to perform a quality test at the end of the card personalization process. In PCA context this ideally would be a complete simulation of the authentication process with the freshly produced card, e.g.:

- requesting a randomized PIP from the card,
- transforming the Polymorphic Identity part of the PIP to an Encrypted Identity for a test service provider,
- decrypting the Encrypted Identity and verifying that the resulting BSN corresponds with the one in the data personalization file.

In this fashion the personalization process would be able to verify the working of the whole authentication process. Actually, the personalization process would also be able to independently verify that BSN-L formed the PIP correctly. However, for this simulation the personalization process requires authentication provider capabilities. This introduces various security, privacy and operational risks and challenges. We will not further elaborate on these but only remark that simulation of the authentication process is not considered acceptable for testing produced cards.

A simple technique to address the above issue is to let BSN-L form a new structure consisting of the PIP, the BSN the PIP is based on and an ECDSA signature binding the PIP and BSN. The card issuer would then be able to validate that BSN-L claims the relation between PIP and BSN. However, the card issuer would not be able to validate the working of the whole authentication process

as in the simulation setup described. The personalization process would neither be able to verify that BSN-L actually formed the PIP correctly. The technique specified in this appendix has these both advantages without the privacy and security disadvantages of the simulation.

E.2 Generation of Verifiable PIP (VPIP)

In the specified technique a *Proof of Conformity* or PoC is added to a PIP, resulting in *Verifiable Polymorphic Identity and Pseudonym* or VPIP. The PoC is specified in Protocol 38 and allows for a proof that a certain identity, i.e. typically a BSN, is correctly embedded in the PI part of the PIP. That is, by using the PoC the correct working of the whole authentication process can be inferred as in the simulation. A PoC consists of two elliptic curve points and 4 integers of l-bits, i.e. a PoC is of size $6l$ bits. The meaning of the PoC elements becomes clear in the proof of Proposition E.1 and is summarized in the discussion following Proposition E.1.

Somewhat complicating in the PoC construction is that PIPs are constructed authentication provider/card issuer specific. For this the Authentication provider Adherence Derived key, i.e. $\mathbf{AA_{Di}}$, is used to embed the identity in a PIP in an authentication provider specific fashion. For simplicity of notation we denote $\mathbf{AA_{Di}}$ by a_i. As indicated in Protocol 39 the verification of a PoC requires the PIP and the *PoC verification public key* that takes the form $V_i = a_i^{-1} \cdot \mathbf{Y}$. We assume that the card issuer is provided an authentic version of the public key V_i from the KMA.

Algorithm 38 below specifies the generation of VPIP structures. In essence Algorithm 38 supplements the original Algorithm 14 with Lines 12 - 19. We let VPIP_{ID} denote the object identifier associated with a VPIP structure. Finally, in implementations the actual string that is embedded as the "identity" in a polymorphic identity will typically be augmented with identity metadata, e.g. with version and type indicators. This means that during VPIP verification the augmented identity should also be used.

Algorithm 38 *GenVPIP*($\text{CI}_{\text{ID}}, Id$) Generate VPIP for CI_{ID} based on Id.

1: SN=SN+1 // increment sequence number.
2: Compute $P_1 = \mathcal{EMB}(Id, m, k, h)$ // OAEP embed Id into curve
3: Compute $P_2 = a_i \cdot P_1$ // make AP specific
4: Compute $Q_1 = \mathcal{W}(\mathbf{IW_M}, Id)$ // keyed wrapping of Id into curve
5: Compute $Q_2 = \mathcal{K}_1(\mathbf{IM_M}, Id) \cdot Q_1$ // additional keyed mapping in curve
6: Compute $Q_3 = a_i \cdot Q_2$ // make AP specific
7: Generate $t \in_R \mathbb{F}_q^*$ and compute $E = \mathcal{MEG}(t, P_2, Q_3, \mathbf{Y}, \mathbf{Z})$ // form base PIP
8: Form byte array $AB_1 = \text{HSM}_{\text{ID}} || T || SN$ // form 16 byte Audit Block
9: $AB_2 = \mathcal{E}_{\text{AES}}(AB_1, \mathbf{SED_a})$ // encrypt AB for Supervisor
10: Compute $Sig = Sig_{\text{dsa}}(\text{PIP}_{\text{ID}} || \text{CI}_{\text{ID}} || E || AB_2, \mathbf{u})$ // sign
11: Form PIP = $(\text{PIP}_{\text{ID}}, \text{CI}_{\text{ID}}, E, AB_2, Sig)$
// end of regular PIP generation
12: Parse E as multi-recipient ElGamal encryption $(Q, R, S, \mathbf{Y}, \mathbf{Z}) \in \mathbb{G}^5$
13: Form $T = a_i^{-1} \cdot R$ // ZK1: proof that T is of this form

14: Generate $k_1 \in_R \mathbb{F}_q^*$
15: Compute $T^1 = k_1 \cdot T$, $V^1 = k_1 \cdot V_i$ and convert to byte arrays $\bar{T^1}, \bar{V^1}$.
16: Compute l-bit bit array $H(\bar{T^1} || \bar{V^1})$ and convert it to integer r_1
17: If $r_1 = 0$ then go to Line 14
18: Compute $s_1 = k_1 + r_1 \cdot a_i \bmod q$.
19: If $s_1 = 0$ then go to Line 14
20: Let $ZP_1 = (r_1, s_1)$
// ZK2: (P, T, V_i) is ElGamal encryption of P_1, i.e. $P, T - P_1$ of form kG, kP_i
21: Generate $k_2 \in_R \mathbb{F}_q^*$
22: Compute $G^2 = k_2 \cdot G$, $V^2 = k_2 \cdot V_i$ and convert to byte arrays $\bar{G^2}, \bar{V^2}$.
23: Compute l-bit bit array $H(\bar{G^2} || \bar{V^2})$ and convert it to integer r_2
24: If $r_2 = 0$ then go to Line 21
25: Compute $s_2 = k_2 + r_2 \cdot t \bmod q$
26: If $s_2 = 0$ then go to Line 21
27: Let $ZP_2 = (r_2, s_2)$
28: Form POC as (P_1, T, ZP_1, ZP_2)
29: Return VPIP $= (\text{VPIP}_{\text{ID}}, \text{PIP}, \text{POC})$.

In Algorithm 39 below we have only taken out the part card personalisation process relevant for PIP verification. This corresponds with lines 17-19 of the PCA card production protocol, i.e. Protocol 6. Lines 2-3, 7-13 of Algorithm 39 are the essential part of the algorithm. In lines 2-3 the PCA card production protocol can validate that the BSN is encoded in the OAEP message that is part of the VPIP. In lines 7-13 the PCA card production protocol can validate that the OAEP message is inside the ElGamal encryption forming the Polymorphic Identity. Together these lines enables the personalisation process to verify that the PIP corresponds with the BSN for which the card is produced.

Algorithm 39 *VerVPIP*$(\text{VPIP}_{\text{ID}}, \text{CI}_{\text{ID}}, Id)$ Verification if $\text{VPIP} = (\text{VPIP}_{\text{ID}}, \text{PIP}, \text{POC})$ is based on Id, i.e. Boolean output.

1: Validate that $(\text{VPIP}_{\text{ID}}, \text{PIP}, \text{POC})$ is a correctly formed VPIP, on failure Return **False** // input validation
2: Validate and parse POC as (P_1, T, ZP_1, ZP_2)
3: Compute $Id' = \mathcal{DEC}(P, m, k, h)$ // OAEP decoding, i.e. Algorithm 10
4: If last step was unsuccessful Return **False** // OAEP decoding failure
5: If $Id' \neq Id$ then Return **False**
6: Parse PIP as PIP, i.e. as $(\text{PIP}_{\text{ID}}, \text{CI}_{\text{ID}}, B_1, B_2, B_3)$
7: Parse B_1 as multi-recipient ElGamal encryption $(Q, R, S, \mathbf{Y}, \mathbf{Z}) \in \mathbb{G}^5$
// verify $a_i \cdot T = R$ and $a_i \cdot V_i = Y$ where latter implicitly defines a_i
8: Parse ZP_1 as (r_1, s_1) with r_1, s_1 non-negative integers
9: Verify that $r_1 \in \{1, 2^l - 1\}$ and $s_1 \in \{1, q - 1\}$, on failure Return **False**
10: Compute $T^1 = s_1 \cdot T - r_1 \cdot R$, $V^1 = s_1 \cdot V_i - r_1 \cdot \mathbf{Y}$
11: if $T^1 = \mathcal{O}$ or $V^1 = \mathcal{O}$ Return **False**
12: Convert T^1, V^1 to byte arrays $\bar{T^1}, \bar{V^1}$
13: Compute l-bit bit array $H(\bar{T^1} || \bar{V^1})$ and convert it to integer v_1
14: If $v_1 \neq r_1$ Return **False**. // end of verification
// verify (Q, T, V_i) is ElGamal encryption of P_1,
// that is: $Q = kG, T - P_1 = kA$ for some k
15: Parse ZP_2 as (r_2, s_2) with r_2, s_2 non-negative integers

16: `Verify that` $r_2 \in \{1, 2^l - 1\}$ `and` $s_2 \in \{1, q-1\}$`, on failure Return` **False**
17: `Compute` $G^2 = s_2 \cdot G - r_2 \cdot Q$, $V^2 = s_2 \cdot V_i - r_2 \cdot (T - P_1)$ `// must be type` kG, kV_i
18: `if` $G^2 = \mathcal{O}$ `or` $V^2 = \mathcal{O}$ `Return` **False**.
19: `Convert` G^2, V^2 `to byte arrays` $\bar{G}^2, \bar{V}^2$.
20: `Compute` l`-bit bit array` $H(\bar{G}^2 || \bar{V}^2)$ `and convert it to integer` v_2
21: `If` $v_2 \neq r_2$ `Return` **False**. `// end of verification`
22: `If` $Ver_{\text{dsa}}(\text{PIP}_{\text{ID}} || \text{CI}_{\text{ID}} || B_1 || B_2 || B_3, \mathbf{U}) =$ **False** `Return` **False**
23: `Return` **True**

Proposition E.1 *With negligible probability of failure Algorithm 39 is successful if and only if the PIP inside $VPIP_{ID}$ is correctly formed by BSN-L and is based on CI_{ID}, Id.*

Proof: We start with the "only if" part of the proposition and assume that the call $VerVPIP(\text{VPIP}_{\text{ID}}, \text{CI}_{\text{ID}}, Id)$ returns **True**. It suffices to prove that, with negligible probability of failure, the PI part of the PIP inside VPIP_{ID} is an ElGamal encryption of $a_i \cdot P_1$ where P_1 is part of the PoC.

To this end, the PoC as specified in Protocol 38 consists of P_1, T, ZP_1, ZP_2. On Line 13 of Protocol 38, T is defined by $T = a_i^{-1} \cdot R$ where R is the second element of the PIP. That is, we have

$$R = a_i \cdot T; Y = a_i \cdot V_i. \tag{28}$$

Here the last part follows from the definition of V_i. This context brings us in the context of Section 4.4. We note that the public key D appearing there is formulated in terms of the basepoint G. However all results also hold when G is replaced by another point, e.g. V_i. By Algorithm 3 from that section it then follows that ZK_1 formed in Lines 12-20 of Algorithm 38 is a zero-knowledge proof of knowledge for Equation (28). Also, Lines 8-14 of Protocol 39 correspond to Algorithm 4 from Section 4.4. That is, if these lines are successful then the verifier knows, with negligible probability of failure, that indeed $R = a_i \cdot T$ and thus that $T = a_i^{-1} \cdot R$ where R is the second element of the provided PIP. In Protocol 38 $(Q, R, \mathbf{Y})$ is formed as an ElGamal encryption of $P_2 = a_i \cdot P_1$. It then simply follows, cf. the proof of Proposition 4.1, that (Q, T, A) is an ElGamal encryption of P_1. Or, equivalently there exists a $k \in_R \mathbb{F}_q^*$ such that

$$Q = k \cdot G; T - P_1 = k \cdot V_i. \tag{29}$$

It actually follows that k equals t from Line 7 of Algorithm 38. From another application of Algorithm 3 it follows that ZK_2 formed in Lines 21-27 of Algorithm 38 is a zero-knowledge proof of knowledge for Equation (29). That is, if Lines 15-21 of Protocol 39 are successful then the verifier knows, with negligible probability of failure, that Equation (29) holds and thus that (Q, T, A) is an ElGamal encryption of P_1. Combined with the first proof of knowledge, the verifier thus knows, with negligible probability of failure, that if Algorithm 39 is successfully concluded $(Q, a_i^{-1} \cdot R, a_i \cdot Y)$ is an ElGamal encryption of P_1. It follows that the verifies knows that (Q, R, Y), i.e. the PI part of the PIP, is an

Elgamal encryption of $a_i \cdot P_1 = P_2$ as desired. By the signature verification in Line 22 the card issuer can infer the PIP originates from BSN-L The "if" part of the proposition is evident from the previous discussion. □

The proof of Proposition E.1 explains the PoC elements (P_1, T, ZP_1, ZP_2):

- P_1 is the "OAEP" embedding in the curve of the identity related to the PIP. That is, decoding P_1 should lead to the identity.
- T is a curve point essentially representing an ElGamal encryption of P_1 under the PoC verification public key V_i,
- ZP_1 is an EC-Schnorr type signature representing a zero-knowledge proof that the previous point is true,
- ZP_2 is an EC-Schnorr type signature representing a zero-knowledge proof linking the plaintext in the PI part of the PIP with T.

We remark that Algorithm 39 fails if either of the two zero-knowledge proofs fail. As indicated in Section 4.4 the probability of failure for these proofs is in the order of 2^{-l}, i.e. 2^{-320} in the context of the Brainpool320r1 curve. We also remark that the PoC size is about six times the size of the group order, i.e. 1.920 bits in case of the Brainpool320r1 curve. We finally remark that from Lines 1-21 of Algorithm 39 the card issuer can infer that the PI part of the PIP is correctly formed regardless of the ECDSA signature verification in Line 22. By this signature verification the card issuer can infer the PIP originates from BSN-L.

F Introducing and discussing nPA (informative)

Broadly speaking two types of national eID schemes exist: centralized (federated) schemes like DigiD and decentralized schemes as for instance used in Germany. In decentralized schemes authenticators (typically a national eID card) are centrally issued but decentralized used: the service provider needs to directly interact with the authenticator to authenticate the user. In a centralized scheme both the issuance as the interaction with the authenticator are done centrally. As decentralized schemes do not deploy central authentication providers, the AP hotspot issue mentioned Section 1.2 does not occur either. According to [2.] this is one of the reasons German government prefers a decentralized scheme: "the use of the card cannot be monitored by government institutions or other parties."

Since 1 November 2010, Germany is issuing contactless eID cards on which the *neue Personalausweis (nPA)* card application resides. The nPA card application allows a user to provide a service provider with a pseudonym (Restricted Identifier) and, if the user consents, optional personal data. The Restricted Identifier pseudonym is service provider specific and the personal data could be name, date of birth, address et cetera. Compare [7]. To facilitate reading from the card the user installs *eID client* software and a card reader, cf. [8]. The eID client

allows to setup communication with the eID card and to handover card communication to the service provider which installs *eID server* for this, cf. [9]. The setup is such that the eID client can technically control what the service provider eID server is reading (or trying to read) from the card. This further discussed below.

nPA is heavily based on electronic passport technology. This technology enables border guards to reliably extract identifying information of citizens from their passport in a contactless fashion, cf. [26]. In nPA this technology is used to extract identifying information from the eID card over the internet, cf. Figure 12. Part 11 of [26] specifies security mechanisms for electronic passports which we now briefly discuss and relate to nPA. To prevent electronic passports being read without user consent, contactless access to the passport is protected. For this two separate protocols (BAC and PACE) exist. Both protocols require the reader to have a low entropy secret information shared by the passport. This information is used to authenticate the reader and to subsequently setup secure messaging. In the passport this secret is derived from data printed *on* the card (Machine Readable Zone or MRZ). PACE is the successor of BAC and is, unlike BAC, resistent against bruteforcing the secret information. The nPA application only allows PACE and uses a PIN, e.g. a 4-6 digit number, as secret shared between the eID card and the eID client. PACE secure messaging is only used locally, i.e. setup between the eID card and the eID client. Prior to the eID server reading data from the eID card, PACE is replaced by EAC (Extended Access Control) secure messaging between the eID card and eID server, see below.

Information residing in the passport is digitally signed through a PKI called Country Signing Certification Authority (CSCA), cf. [26, Part 12]. This protection is called Passive Authentication (PA). Through this PKI a passport is typically also equipped with a certified private signing key securely stored in the chip. This also allows the passport to prove it is authenticate through the CA protocol), i.e. that it is not cloned. It allows the passport to show possession of a private key whose public key is certified ('signed') through PA. The CA protocol works indirectly: secure messaging is setup based on the Diffie-Hellman protocol based on a ephemeral key generated by the reader and a fixed, PA certified key from the passport. If the reader can successfully read data from the passport over the setup secure messaging, the passport is considered authentic.

Another standard [4] enables authorized border guards to securely extract fingerprint data from electronic passports thereby allowing biometric verification of the passport holder. This is secured with a protocol called *Terminal Authentication (TA)* that is additional to the protocols in [26, Part 11]. This uses card verifiable TA certificates (also known as inspection certificates) implemented through another PKI infrastructure called Country Verifying Certification Authority (CVCA): border control systems possess a private key bound to a TA certificate issued under the CVCA by a intermediary authority called Document Verifying Certification Authority (DVCA). These certificates are verifiable by the electronic passport ('card verifiable') allowing it to decide if extracting the fingerprints is authorized or not. The TA protocol is coupled with the CA pro-

tocol and the combination is known as *Extended Access Control (EAC)*. EAC allows mutually authenticated end-to-end secure messaging between reader and passport protecting both confidentiality and integrity of data exchanged. As passports cannot handle Certificate Revocation Lists, TA certificates are short-lived. Moreover, passports keep track of time by using the issue dates of seen TA certificates. We note that two different versions of EAC exist corresponding with the order of the CA and TA protocol. In EACv1 [4] the order is CA followed by TA whereas in EACv2 [5] the order is reversed.

The RI pseudonyms are service provider specific, i.e. different service providers get different pseudonyms that are not cryptographically linkable. That is, the pseudonyms do not allow service providers to decide if they correspond to the same person. Of course additional personal information provided could allow for this, implying one should exercise restraint in providing this. The nPA application is equipped with a CA public key to prove authenticity. If each nPA instance would have a unique CA public key this would allow indirect identification of the card by the service provider. As a remediation, batches of German eID cards share a CA public/private key pair. To further facilitate this the cards in a batch are all signed with one document signer certificate. At batch initiation, a fresh document signer and CA key pair is generated. The document signer public key is bound in a digital certificate and the document signer private key is used to sign the CA public key. The resulting package (CA public/private key, document signer signature, document signer certifiate) is then use throughout the batch; the document signer private key can be destroyed.

The eIDAS regulation [18,19] also requires that timely revocation of authenticators must be possible. This is also supported in nPA but it is somewhat complicated due to the pseudonymization. For this a central government organization distributes revocation lists ('black lists') that contain the pseudonyms corresponding with revoked eID cards. As these lists are service provider-specific, the number of lists grows linearly with the number of service providers, making this mechanism somewhat cumbersome. This especially holds when one wants to have a short refresh period for these lists, e.g. four hours. Through these revocation lists also privacy related data can leak. Indeed, if there is only one person that has revoked its card in a refresh period, then only his pseudonyms will be new on the lists allowing linking. This is another example that reliability and privacy can be conflicting: for the first property one wants short refresh periods and for the second long refresh periods.

Next to black lists there also exist 'white lists' in nPA. These lists are also service provider-specific and contain all pseudonyms corresponding to issued eID cards. The white lists will be distributed in the (unlikely) event that one of the shared CA public keys gets compromised, e.g. through a novel attack. In this event, an attacker can introduce rogue RI pseudonyms, i.e. non-existing ones. Deploying these lists will not only present a logistic challenge but also a communicational one: it seems questionable if German government can convince its citizens to keep on using the card in case of this event. Indeed, in this event

an attacker can also impersonate existing users if he knows their RI pseudonyms at service providers.

In essence, nPA is a re-use of the passport PACE, PA, EAC mechanisms. In essence its working is as follows, cf. Figure 12:

1. A user wants to login to a service provider web service and consents in data to be read from the card (RI pseudonym and certain personal data).
2. The user places its eID card on a reader, enters his PIN which allows the eID client to communicate with the card using the PIN based PACE protocol.
3. The service provider sends its TA certificate to the card and sets up EAC based secure messaging with the card thereby using the certified CA public key read from the card.
4. The user is informed by the eID client on the identity of the service provider (taken from the TA certificate) and the data it wants to read from the card.
5. Only if the user consents, the eID client allows the service provider to read the data from the card.
6. The service provider runs the Restricted Identification (RI) protocol reading an RI pseudonym from the card. Additionally the service provider reads data from the card. The eID client enforces that the user consents that this data is read.
7. The service provider checks if the RI pseudonym is on the black list, if this is the case the authentication is rejected.
8. Optionally, e.g., in case of compromise of shared CA keys, the service provider checks if the RI pseudonym is on the white list, if not the authentication is rejected.

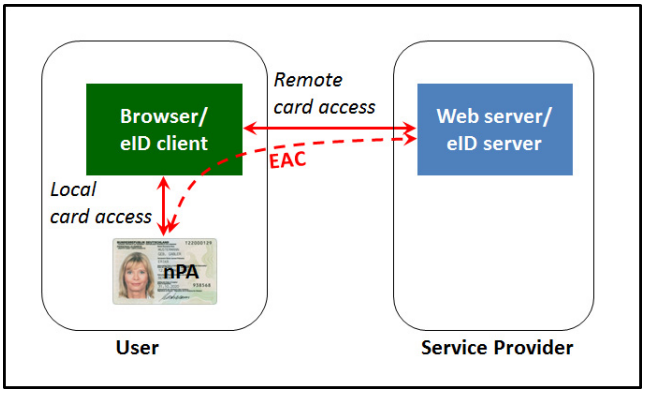

Figure 12. nPA (German eID card)

We mentioned earlier that EAC secure messaging is end-to-end implying that the eID client acts as a proxy. The eID client enforcement indicated in Step

6 is possible as EAC secure messaging does not fully encrypt the commands (Application Protocol Data Unit command or APDU-C) sent to the card and responses (Application Protocol Data Unit response or APDU-R). In APDU-C the 4 byte command instruction ("Header") and in the APDU-R the two byte error codes ("status words") are not encrypted. EAC secure messaging does protect the integrity of the whole messages, including command instructions and error codes. Compare [6, Figures 4,5]. In this way, the eID client can detect that the service provider wants to read certain data from the card but has no access to the actual data retrieved.

Since its introduction in 2010 nPA is not yet broadly used. According to [24] only five percent of German citizens use nPA in the summer of 2015. As one of the reasons the lack of service providers using nPA is mentioned. The complexity for service providers to directly communicate with nPA might be one of the reasons for this. This is also indicated by the introduction of the eID service concept by German government in 2012. Through this concept a service provider can interact with nPA through a SAML [41] based service implemented in an outsourced eID-server [9] called *eID Service*. German government is actively stimulating the use of eID Services by mentioning many (10) potential eID service providers on the website [10]. This number also indicates that German government anticipates that many service providers (will) use the eID service.

The eID service also suggests that German government is reconsidering the earlier mentioned requirement that the use of the card cannot be monitored. Indeed, the eID service has access to all RI pseudonyms and to all personal data read from the nPA allowing pseudonym linking. Moreover the mentioned linkability risk related to the revocation lists is likelier to become manifest at an eID service. We conclude that the original nPA design whereby the service provider directly interacts with the eID card has nice privacy properties but these are significantly reduced when nPA is used in a centralized fashion, i.e. through the eID service concept.

In a cooperation between the German and French government, the nPA specification has been included in a broader specification called the eIDAS token, cf. [5,6,7]. In this specification a third version of Extended Access Control is specified as well as an alternative for the Restricted Identification protocol. Moreover, further support for attributes is specified including writing of attributes on the card. For the description of PCA these extensions are not relevant.

G PCA indistinguishability (informative)

Theoretically one can implement absolute anonymity during PCA authentication. That is, one can ensure that *all* issued PCA instances are cryptographically indistinguishable. For instance, we could deploy PCA without shared Chip Authentication public keys used for clone detection and fully rely on the Status Service for this. By doing so we would further increase reliance on the confidentiality of the polymorphic forms placed within PCA. Indeed, without Chip Authentication possession of such a polymorphic form would allow authentication for the corresponding user. Consequently, this would require further reliance of the handling of these polymorphic forms at the PCA authentication provider, issuer and at the Activation service where these forms are generated. One could also replace the shared Chip Authentication public keys with a group based signature scheme [12]. However, this would have serious impact on the complexity and performance of PCA. Finally, as using shared Chip Authentication public keys is also stipulated within the German eID Token specification [5] we think that deviating from this is not good practice.

If absolute indistinguishable during PCA authentications is practically infeasible, the question arises "how much" distinguishable is acceptable. Or in more practical terms: suppose a party has full read access to PCA instances, i.e. being able to retrieve random polymorphic forms. What minimal group size k of PCA instances is acceptable the party can assess a given PCA instance belongs to? Or alternatively formulated: what level of k-anonymity sensu [47] is required in PCA practice?

We will argue that a group size k of 20.000 is consistent with current international and Dutch good practice. The number of 20.000 is based on the US government standards for individually identifiable health information [14, Section 164.514]. These standards include a method for de-identification of personal data, i.e. ensuring that resulting data cannot be traced back to an individual. Part of this method consists of the removal of postal codes with a population of 20.000 or less and the removal of ages less than 90 years. As the age of the PCA holder is not available in PCA, choosing a PCA distinguishability number of 20.000 is substantial stricter requirement than [14]. The Dutch Data Protection Authority (DPA) supervises various medical services processing "non-identifiable" data [15]. In that practice, cf. `https://www.zorgttp.nl`, it is commonly accepted that such data may contain the four letters of a Dutch postal code and the age of the individual. In many cases Dutch four letter postal code have populations less than 20.000. We conclude that a PCA distinguishability number k of 20.000 is far stricter that what is required in both US standards and Dutch practice on non-identifiability of personal health information. We therefore think that this choice is acceptable.

H Extensions and applications (informative)

In this section we discuss extensions and possible specific applications of the building blocks specified in the previous sections. The purpose of this discussion

is only to elaborate on the innovative potential of these building blocks. This discussion does not imply intension to implement these building blocks.

H.1 EI/EP as self-contained, legally binding assertion holders

Algorithms 17, 18, 19 and 20 specify the EI/EP generation algorithms including the forming of the EC-Schnorr signature. The AP can conveniently augment an EI/EP with additional data in the EC-Scnorrr signature scope formed in these algorithms. This additional data can then be validated in augmented versions of Algorithms 21 and 23. As these signature are HSM formed, they can provide a strong mechanism to convey AP attributes and user consent to the service provider. The archived EI including its EC-Schnorr signature, would actually allow the service provider to irrefutably prove this user consent. See also Appendix H.2 where we relate this to the European General Data Protection Regulation [17]. Possible applications include contract signing and change of fundamental data, e.g., bank account number, donor codicil data, authorizations. Such "expressions of will" could also be part of a user re-authentication initiated by the service provider. For instance, as part of a bank account number change at a tax authority the user is redirected back to the authentication provider for re-authentication. The authentication request also includes an appropriate statement to be presented to and approved by the user. If successful the authentication response contains a signed approval of the bank account number change.

This setup also naturally fits in the *remote signing* context of the eIDAS regulation [18]. Remote signing was one of the main differences between the eIDAS regulation and the electronic signature Directive 1999/93/EC it replaces. This support is provided by relaxing the definition of an advanced electronic signature. The 1999 directive stipulates that such a signature is "created using means that the signatory can maintain under his sole control". However the eIDAS regulation [18, Article 26] requires that such a signature is "created using electronic signature creation data that the signatory can, with a high level of confidence, use under his sole control". The term "high" here refers to the level of authentication implying that an advanced electronic signature can be formed by an authentication provider whereby the user authenticates at this level. In European context one can thus argue that if the used authenticator is of eIDAS level High that then an augmented EI/EP constitutes "electronic signature creation data" as stipulated above. This would mean that an augmented EI/EP can argued to be legally binding. Also, augmented EI/EP could conveniently form the basis for qualified remote signing services sensu the eIDAS regulation. We note that EC-Schnorr is an European approved signature algorithm, cf. [21].

We finally observe that Algorithms 17, 18, 19 and20 also allow to additionally encrypt data using the ElGamal public key included. This data can also be placed within the scope of the EI/EP signature. In doing so, an authentication provider has a self-contained mechanism to convey attributes to service providers in a manner protection both confidentiality and authenticity.

H.2 Privacy friendly data exchange

Consider two service providers $\mathrm{SP}_1, \mathrm{SP}_2$ identifying users with pseudonyms. Suppose that SP_1 posseses an encrypted pseudonym $\mathrm{EP@SP}_2$ at SP_2 of one of its users, i.e. coupled with its pseudonym at SP_1. Then this allows SP_1 to request data D of the user from SP_2. Indeed, SP_1 queries SP_2 referring to $\mathrm{EP@SP}_2$, then SP_2 can deduce the user pseudonym from this, collects the appropriate D and sends this to SP_1. The latter would be able to register this data under the SP_1 user pseudonym. We note that $\mathrm{EP@SP}_2$ is only a reference and does not reveal SP_1 the user pseudonym at SP_2. In particular, if there are more service providers like SP_1 pulling data from SP_2, then by the EP generation algorithms all would get different, cryptograpicaly unrelatable EPs at SP_2. That is, based on the EPs alone these service providers would be able to link users. In the described use case, SP_1 pulls data from SP_2. We can also support another use case whereby SP_2 periodically pushes (sends) SP_2 user data to SP_1 using $\mathrm{EP@SP}_1$.

There needs to be a legal basis for such exchanges. Compare Article 6 of the European General Data Protection Regulation [17] where several such bases are specified. A prominent basis is a user ("data subject") giving consent for the exchange. As stipulated in [17, Article 7] both $\mathrm{SP}_1, \mathrm{SP}_2$ need to be able to demonstrate this consent. In this context the assertion properties of EIs/EPs indicated in Appendix H.1 are convenient. Indeed, we can allow a user to authenticate to SP_1 whereby this user consent for pulling/pushing data is asked by the authentication provider. If given, the authentication provider then generates an assertion $\{\mathrm{M_C}, \mathrm{EP@SP}_2\}$ for SP_1 whereby $\mathrm{M_C}$ constitutes a user consent message incorporated in $\mathrm{EP@SP}_2$, i.e. part of the EC-Schnorr signature in $\mathrm{EP@SP}_2$. For instance, in the first use case $\mathrm{M_C}$ could be a statement as "data subject allows SP_2 data of type X to be pulled by SP_1 in the time period Y". In the second use case $\mathrm{M_C}$ could be a statement as "data subject allows data of type X to be pushed by SP_2 to SP_1 in the time period Y". In line with EI/EP generation such assertions would be generated by the authentication provider HSM.

For data quality reasons it is better if the assertions also refers to the pseudonym of the user at the other service provider, i.e. to $\mathrm{EP@SP}_1$ in the first use case. For instance, if SP_2 discovers an incorrectness in the data pulled by SP_1 then it should be possible for SP_2 to indicate SP_1 the user it pertains to. Even more important is a dispute situation between the user and SP_1 on the data pulled from SP_2. Then SP_1, SP_2 together should be able to demonstrate that the data is correctly linked (also deploying the algorithms from Section 4.4). Note that [17, Article 16] also stipulates the right to rectification.

To avoid having SP_1 always sending $\mathrm{EP@SP}_1$ to SP_2 we only incorporate a hash of this encrypted pseudonym also to be signed in the $\mathrm{EP@SP}_2$. We identify two kinds of *link assertions*:

$$L_{\mathrm{p@1}\to\mathrm{p@2}} = \{\mathrm{M_C}, \mathcal{H}(\mathrm{EP@SP}_1), \mathrm{EP@SP}_2\};$$
$$L_{\mathrm{p@2}\to\mathrm{p@1}} = \{\mathrm{M_C}, \mathcal{H}(\mathrm{EP@SP}_2), \mathrm{EP@SP}_1\}.$$

With $L_{\mathrm{p@1}\to\mathrm{p@2}}$ the service provider SP_1 is able/allowed to pseudonymously pull data from SP_2 or push data to SP_2. A similar property holds for $L_{\mathrm{p@2}\to\mathrm{p@1}}$. Note

that we let the arrow direction correspond with the push direction. In the link assertions, M_C corresponds with a consent message of the user. Also M_C and the hash value are incorporated in the EC-Schnorr signature of the encrypted pseudonym. If SP_1 possesses $L_{p@1\to p@2}$ and SP_2 possesses $L_{p@2\to p@1}$ then full data exchange between SP_1, SP_2 is possible. Of course, if SP_1 cooperates then technically SP_2 is able to push data to/pull data from SP_1 using $L_{p@1\to p@2}$. Indeed, SP_1 could use $\mathcal{H}(EP@SP_1)$ to lookup the user. In this sense, taking the hash is more of formal/legal value then of cryptographical value: we need to trust the service providers to abide by the user consent given.

Clearly one can similarly base the exchange on encrypted identities instead of encrypted pseudonyms. This leads to to similar expressions, e.g. $L_{i@1\to p@2}$. Even in the situation that both service providers are allows to process the user identity (BSN) this still has the added value of the explicit consent recorded. Indeed, there might be situations where exchanges between public service providers are not governed by law but by user consent. Particulary interesting is the situation of data exchange between public and private parties. The first type of parties process BSNs which is not allowed for the second type of parties. By combining an EI and EP we can also facilitate data exchange in that context. This situation occurs in health care. There is a trend of empowering patients by letting them be in control of their own medical data. The "MedMij" (Medical Me) standardization[4] facilitates this by enabling patients to collect and control all their medical data at a private "MedMij" web environment. This facilitates patients to share their medical data with other doctors and to ask for second opinions. Within "MedMij" it should be possible to exchange data from health care providers (hospitals, general practitioners) to a private "MedMij" web environment. Health care providers register user data using the BSN which is not allowed to processed by the "MedMij" environment. Clearly the discussed usage of EI/EP conveniently allows exchange. For completeness we have further specified this mechanism into two protocol pairs exchanging data between a hospital H and a "MedMij" environment MM.

The first pair deals with data being pulled by MM and the second pair deals with data being pushed by H. These protocols are depicted in Figure 13 and Figure 14 respectively. Each protocol pair starts with a link activation whereby the user (patient) consents to the exchange of his medical data from H to MM. This also provides MM with the required link assertion. These protocols are similar to Protocol 3. The authentication protocols could also incorporate notification of the exchange to the user inspection service similar to Lines 9-12 in Protocol 1. The second protocol in each pair deals with the actual exchange. Of course many other variants can be designed using these principles. The protocols are "happy flow" leaving out systematic error handling and also leave out possible log writing.

Protocol 13 *User pull activation at MM*
User activates data pull from H to MM through authentication provider.

1: `User wants MM pulling his data from H`

[4] www.medmij.nl

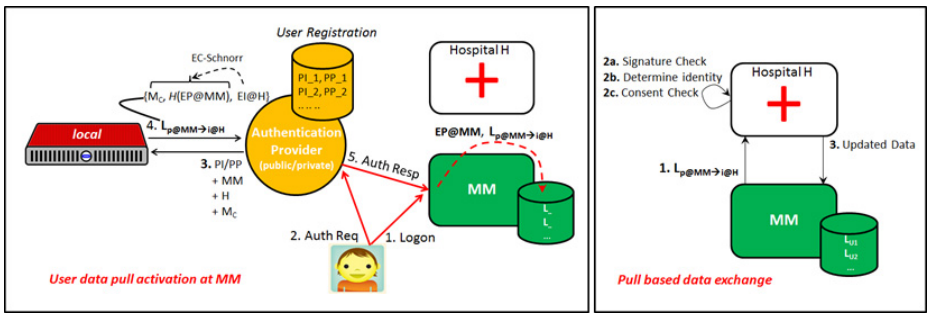

Figure 13. "Medical Me" pull based activation and data exchange

2: MM directs user to AP with pull link request by MM from H
3: User authenticates to AP using authenticator activated in Protocol 1
4: User consents to AP forming link assertion allowing MM pull data from H
5: AP selects user PI+PP and forms EP@MM and link assertion
$L_{\mathtt{p@MM\to i@H}} = \{\mathtt{M_C}, \mathcal{H}(\mathtt{EP@MM}), \mathtt{EI@H}\}$; by augmented Algorithms 17, 19
6: User is redirected to MM with EP@MM and link assertion $L_{\mathtt{p@MM\to i@H}}$ in response
7: MM determines the user pseudonym from EP@MM by Algorithm 23
8: On success, user is activated under his pseudonym and $L_{\mathtt{p@MM\to i@H}}$ at MM

Protocol 14 *Data pull by MM from H*
MM periodically pulls medical data from H for activated users.

1: MM periodically performs the following
2: **for** all activated users **do**
3: MM pulls H for updates referring to $L_{\mathtt{p@MM\to i@H}}$
4: H validates $L_{\mathtt{p@MM\to i@H}}$ using augmented version of Algorithm 21
5: If successful H also determines identity I from EI@H else return Error
6: Collect all updated data for I and return this to MM
7: MM updates data for user
8: **end for**

Protocol 15 *User push activation at H*
User activates data push from H to MM through authentication provider.

1: User U wants H pushing his data to H
2: H directs U to AP with link request between MM and H
3: User authenticates to AP using authenticator activated in Protocol 1
4: User consents to AP providing link assertion allowing H push data to MM.
5: AP selects user PI+PP and forms EI@H and link assertion
$L_{\mathtt{i@H\to p@MM}} = \{\mathtt{M_C}, \mathcal{H}(\mathtt{EI@H}), \mathtt{EP@MM}\}$; by augmented Algorithms 17, 19
6: User is redirected to H with EI@H and link assertion $L_{\mathtt{i@H\to p@MM}}$ in response
7: H determines the user identity from EI@H by Algorithm 21
8: On success, user is activated under his identity and $L_{\mathtt{i@H\to p@MM}}$ at H

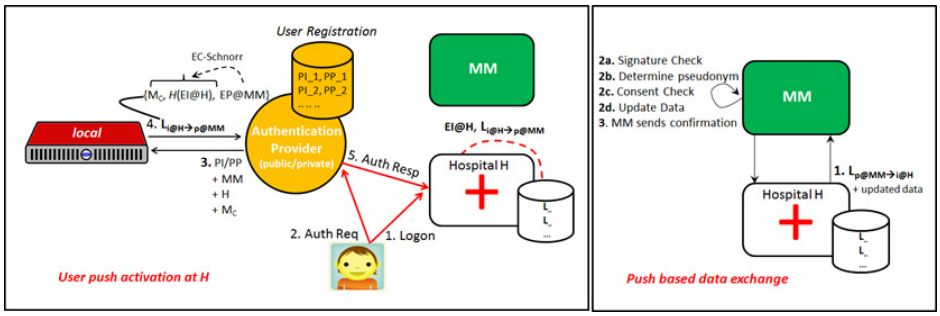

Figure 14. "Medical Me" push based activation and data exchange

Protocol 16 *Data pull by MM from H*
H periodically push medical data to H for activated users.

1: `H periodically performs the following`
2: **for** all activated users **do**
3: `Collects all updated data for I`
4: `Pushes updated data to MM referring to` $L_{\texttt{i@H}\rightarrow\texttt{p@MM}}$
5: `MM validates` $L_{\texttt{i@H}\rightarrow\texttt{p@MM}}$ `using augmented version of Algorithm 21`
6: `If successful MM determines pseudonym using EP@MM else return Error`
7: `MM updates data for user and confirms this to H`
8: **end for**

H.3 Polymorphic eID scheme voting as alternative for postal voting

H.3.1 Vulnerabilities in current postal voting

During Dutch elections a group of Dutch citizens exists that we shall refer to as "voters abroad". These are persons that have voting right but temporarily reside abroad. Expats are an example of such persons. As they live abroad they cannot vote in one of the regular polling stations. To facilitate voting for these persons they can vote by mail (postal voting). Compare https://www.denhaag.nl. A voter abroad can register as a "Dutch voter permanently living abroad" at the city of The Hague. This city maintains the national Register Voters Abroad (RVA). A voter abroad has to fill in a form and to mail this together with a copy of a Dutch passport or identity card to the city of The Hague. In the form the citizen can also indicate if he wants to receive the ballot paper by post or at e-mail address indicated in the form.

Before the city of The Hague can send ballots to persons registered in RVA, the candidate list has to be finalized. This complicates sending ballots well in advance to the voter abroad. However, the city sends a "voting ticket" to the voter abroad well in advance by mail. At the voting date, RVA sends voters abroad a ballot by mail or by e-mail if the voter abroad chose for that. In the

latter case the voter has to print out the ballot himself which will be faster than receiving the ballot through regular mail. The voter abroad then completes the ballot and puts it in an envelope together with the voting ticket and a copy of a Dutch passport or identity card. The envelope then needs to be mailed to the postal polling station of the city of The Hague. At this station, the envelope is opened and some validations are conducted on its contents including on the copy of the Dutch passport or identity card. If successful, the ballot is placed into a ballot box further following the regular voting process.

In the postal voting process, the following weaknesses can be identified:

A. Postal dependency
The voting process is highly dependent of the postal delivery of the ballot and other records to and from abroad. Various reports have appeared in the media that sometimes ballots from abroad were not received on time and consequently not included in the election results.

B. Weak registration process
In principle a fraudster can register at RVA and vote on behalf of a voter abroad with a copy of his Dutch passport or identity card.

C. Breach of voting secrecy
The employee of the postal polling station gets both access to the identity of the voter abroad and the vote he cast.

Especially the first weakness gets media attention and stimulates the city of The Hague to come with a better alternative. For this, the city is considering an online alternative. When possible the alternative process should also address the other two weaknesses.

We remark that online voting introduces three generic weaknesses:

1. **Distributed Denial of Service (DDOS) attacks at voting site**
The voting process could be disrupted when DDOS attacks are conducted at the voting site, making it impossible to vote.
2. **Vote selling/ coercion (no voting freedom)**
During online voting there exists no publically accessible polling station that can ensure that a voter can freely vote. This implies that a voter can cast its vote under direct control of somebody else allowing vote selling or being forced to cast another vote then intended.
3. **Lack of control and observability**
A regular Dutch voting process is publically observable: anybody is allowed to observe the process in a polling station, the counting of the ballots and the filing of the voting result. The lack of a physical polling stating in online voting makes its intrinsically more difficult to be pubically observable.

As far as possible, these weaknesses should be addressed in an online alternative for postal voting. The last two weaknesses also occur in postal voting. Mitigation of these in an online alternative would therefore be beneficial.

H.3.2 Basic polymorphic eID setup

In this section we describe a basic polymorphic eID scheme setup. This follows the current postal voting setup as much as possible and is also kept as simple as possible. We remark that several variants and enhancements exist. The latter we discuss in Appendix H.3.4 below. Within the online alternative similar phases occur as within postal voting:
- registration as voter abroad,
- voting preparation,
- voting website establishment,
- online voting,
- determination of voting result.

Each of these phases is described below in the form of a protocol. The protocols are "happy flow" leaving out systematic error handling and also leave out possible log writing.

Registration as voter abroad
The voter abroad needs to register online at the RVA website maintained by the city of The Hague. During registration the user also also provides an e-mail address that will be used to notify the voter on the upcoming election. Clearly, one can also use other notification mechanisms such as SMS. The RVA registration can be done online and can be based on any authenticator within the polymorphic eID scheme. This process is specified in Protocol 17 and illustrated in Figure 15

Protocol 17 *Registration as voter abroad at the RVA website*

```
1: User wants to register as voter abroad at the RVA website
2: User authenticates to RVA website through AP providing EI by Protocol 2
3: User registers as voter abroad indicating a notification email address
4: The RVA website performs some verifications on the request // user has
   Dutch nationality?
5: If the verifications fail, the protocol returns an error to the user
6: User re-authenticates to RVA website through AP providing EI explicitly
   confirming registration, cf. Appendix H.1              // optional step
```

The confirmation in Line 6 of Protocol 17 incorporates additional text within the EC-Schnorr signature of the EI as indicated in Appendix H.1. This text could include the notification email and a statement pertaining that the user wants to registered as voter abroad starting from a certain date. This could help to resolve possible later disputes between the user and the city of The Hague.

Voting site preparation
In preparation of the election, a voting website (VW) is setup with a URL reflecting the upcoming elections, e.g. `www.TK2021.nl`. The administration of this web site is placed at an online polling station separated from the voter abroad registration. That is, there are no persons having access to both the website and the voter abroad registration. For the actual establishment of the voting website, the online polling station waits until the candidate list is finalized.

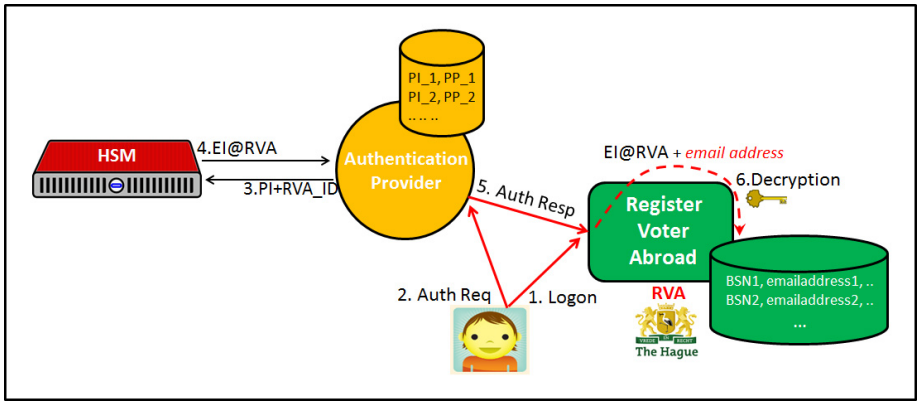

Figure 15. Registration as voter abroad at RVA

Voting website establishment

When the candidate list is finalized and the paper ballots are being printed, also the voting website is also established. This is done in three steps. In the first step, the website employees request EP decryption and closing keys, i.e. $\mathbf{PD_{Pi}}$, $\mathbf{PC_{Di}}$, as well as a DEP receiving key, i.e. $\mathbf{DR_{Di,R}}$, from the KMA. With this keys it is possible:

- for the voting website to extract pseudonyms from a DEP@VW,
- for voters abroad to pseudonymously authenticate to the voting website.

The second step consists of a file F being generated by RVA in cooperation with BSN-L. In the third step this file F is sent by RVA to the voting website administration allowing it to establish the voting website. Steps 2 and 3 are specified in Protocols 18, 19 respectively and illustrated in Figure 16.

Protocol 18 *Forming of file F by RVA*

1: `RVA initiates new files` F `and` S.
2: **for** all citizens registered as voter abroad **do**
3: `RVA sends` Id `(BSN) and other data` D `to BSN-L requesting DEP@VW`
4: `BSN-L validates` Id `in combination with` D
5: `On failure, a failure reason is returned to RVA`
6: `On success, BSN-L returns DEP@VW to RVA generated by Algorithm 16`
7: `RVA writes failures to file` F `and successes to file` S
8: **end for**
9: `RVA lexicographically orders the DEPs in` F

Line 9 of Protocol 18 ensures that any implicit information arising from the order in which RVA requests the DEPs in Lines 2-8 is removed from file F. It also removes the relation between the RVA registration and file F. That is, a DEP in file F can by itself not be linked to a user in RVA. For further correlation protection, one could let BSN-L generate the complete (lexicographically ordered) file F. File F is sent by RVA to the voting website which then processes it as indicated in Protocol 19.

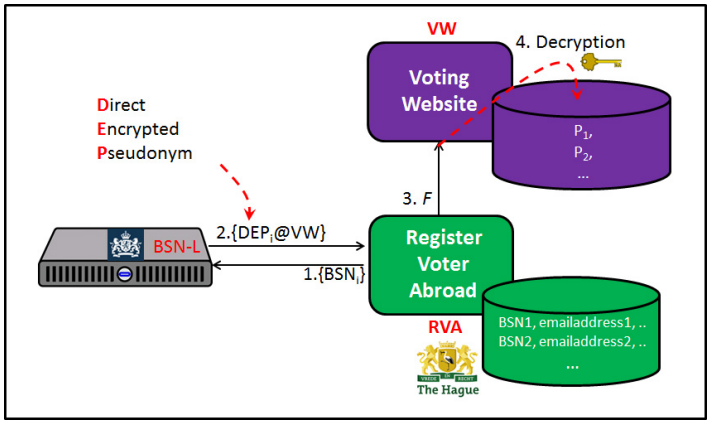

Figure 16. Voting website establishment

Protocol 19 *Initiation of voting website VW*

1: `RVA forms file` F `using Protocol 18 and sends this to VW`
2: `VW receives file` F `// Process of file` F `by VW`
3: **for** all DEPs in file F **do**
4: `VW determines pseudonym P from DEP@VW by Algorithm 24`
5: `VW registers user as voter abroad under P`
6: **end for**
7: `VW deletes file` F `and notifies RVA on successful processing of file` F
8: `RVA deletes file` F

We already arranged by lexicographically ordering F that it cannot be related to the RVA registration. The deletion of file F in Lines 7,8 of Protocol 19 further removes correlation of users between RVA and VW. By letting RVA wait deleting file F until the voting website has processed it, allows RVA resending it.

Online voting by voter abroad

At the election start, the RVA sends notifications to the voters abroad using the registered email. This email contains a reference to the voting site. This setup might requires some attention, to prevent possible abuse of this setup by phishers during the election. On the other hand, the group of voters abroad is probably too small to be of interest for phishers. In the happy flow, the voter abroad then pseudonymously authenticates at the voting site. The election result is stored in the voting site in an *election result table (ERT)* essentially consisting of two columns: the user pseudonym and the casted vote. This table grows in size during the election. The election process is further specified in Protocol 20 and illustrated in Figure 17.

Protocol 20 *Online voting by voter abroad at voting website VW*

1: `User connects to online voting website VW`

2: User authenticates under pseudonym P to VW through AP using Protocol 3
3: VW verifies if P is registered as voter abroad, cf. Protocol 19
4: If P is not registered, then the user is directed to the RVA website
5: Otherwise VW verifies if P already cast a ballot by inspecting ERT
6: If P already cast a ballot, the user is informed on this without revealing the ballot value
7: Otherwise the user is allowed to cast a ballot B
8: VW updates ERT with a row $[P, B]$

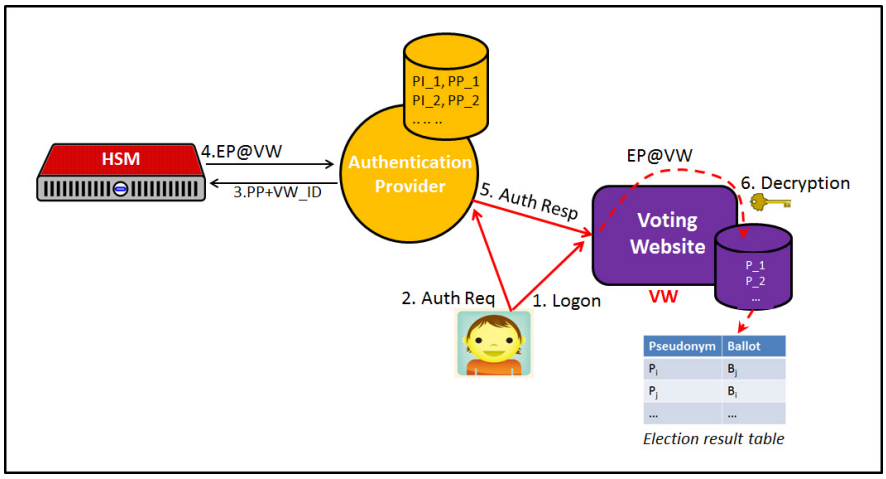

Figure 17. Online voting by voter abroad

Determination of online election result

When the election is closed the online polling station counts the ballots and creates a report on the election result for the online polling station. This is comparable with the role of a regular polling station. The basis for the report is the election result table formed in the voting website. This table is exported from the voting website by the employees of the online polling station. In this export the pseudonyms are replaced with their hashes. The hashed election result table allows determining the election result. This process is indicated in Figure 18.

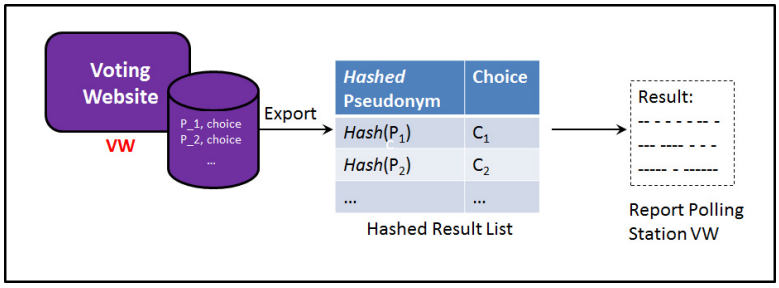

Figure 18. Determination of online election result

The polymorphic eID scheme has implemented both technical and organizational controls preventing a single participant to determine the user identity from a pseudonym. Hashing of the pseudonyms further strengthens these properties. Hashing of pseudonyms is not routinely done in the polymorphic eID scheme as it hampers re-pseudonymisation, i.e. change of closing key, and scheme key roll-over. In the online voting use case this is not a concern as the voting website is used only once.

H.3.3 Comparison with vulnerabilities in postal voting

Below we discuss the three weaknesses identified in postal voting (Appendix H.3.1) with the online setup. We compare the online setup with the setup based on

A. Postal dependency
The process is completely online and no longer dependent of postal delivery.

B. Weak registration process
The voter abroad registration process is now online and based on eIDAS authentication level Substantial or higher. This is significantly stronger than the current postal registration process used in postal voting.

C. Breach of voting secrecy
No longer persons exist that have access to both the identity of the voter abroad and the ballot cast. More specifically:

- RVA employees have access to the identities of the voters abroad but not to the ballots they cast,
- employees of the online polling station have access to the ballots cast but not to the identities of the voters abroad.

We further note that RVA employees are not able to extract the voting website pseudonyms from the Direct Encrypted Pseudonyms in file F, cf. Protocol 18. In this fashion, the polymorphic eID setup even offers protection against collusion between RVA and online polling station employees.

H.3.4 Mitigation of generic weaknesses in online voting

The described basic online voting process for voters abroad can be enhanced to mitigate the three generic weaknesses in online voting mentioned at the end of Appendix H.3.1:

1. **Distributed Denial of Service (DDOS) attacks at voting site**
   The basis for mitigation will be regular DDOS controls an organization can take. These include having ample bandwidth and the deployment of a "network cleansing service" close to the internet backbone that can early distinguish good and bad network traffic. One can also arrange that the notification email sent by RVA implements a segregation between legitimate voters abroad and others.
2. **Vote selling/ coercion (no voting freedom)**
   This weakness can be mitigated by giving voters abroad the possibility to cast their ballots several times whereby only the last ballot cast counts. In this way, a voter abroad can correct an earlier ballot that was cast under coercion. We note that this setup is also chosen in the Estonian online election process.
3. **Lack of control and observability**
   Mitigation can be based by publicizing the hashed election result table and providing voters abroad the hash of their pseudonym as part of the online voting process. This hash would then function as an "election receipt". This receipt enables the voter to verify that his ballot was part of the hashed election result table and thus of the election result. This construction was also part of the Rijnland Internet Election System (RIES) used by Dutch water authorities in 2006[5]. This construction conflicts with the freedom to vote. Indeed, with an election receipt, a voter abroad can also "prove" what he has voted. However, in our context simple constructions are possible where the voter abroad can verify that his ballot was part of the election result without being able to "prove" this to others.

[5] Compare https://en.wikipedia.org/wiki/Rijnland_Internet_Election_System.