# Polymorphic Encryption and Pseudonymisation for Personalised Healthcare

A Whitepaper

Version 1.0

Eric Verheul, Bart Jacobs, Carlo Meijer,
Mireille Hildebrandt, Joeri de Ruiter

Institute for Computing and Information Sciences
Radboud University Nijmegen, The Netherlands

April 26, 2016

## Summary

Polymorphic encryption and Pseudonymisation, abbreviated as PEP, form a novel approach for the management of sensitive personal data, especially in health care. Traditional encryption is rather rigid: once encrypted, only one key can be used to decrypt the data. This rigidity is becoming an every greater problem in the context of big data analytics, where different parties who wish to investigate part of an encrypted data set all need the one key for decryption.

Polymorphic encryption is a new cryptographic technique that solves these problems. Together with the associated technique of polymorphic pseudonymisation new security and privacy guarantees can be given which are essential in areas such as (personalised) healthcare, medical data collection via self-measurement apps, and more generally in privacy-friendly identity management and data analytics.

The key ideas of polymorphic encryption are:

1. Directly after generation, data can be encrypted in a 'polymorphic' manner and stored at a (cloud) storage facility in such a way that the storage provider cannot get access. Crucially, there is no need to a priori fix who gets to see the data, so that the data can immediately be protected.

   For instance a PEP-enabled self-measurement device will store all its measurement data in polymorphically encrypted form in a back-end data base.

2. Later on it can be decided who can decrypt the data. This decision will be made on the basis of a policy, in which the data subject should play a key role.

   The user of the PEP-enabled device can, for instance, decide that doctors $X, Y, Z$ may at some stage decrypt to use the data in their diagnosis, or medical researcher groups $A, B, C$ may use it for their investigations, or third parties $U, V, W$ may use it for additional services, *etc.*

3. This 'tweaking' of the encrypted data to make it decryptable by a specific party can be done in a blind manner. It will have to be done by a trusted party who knows how to tweak the ciphertext for whom.

This PEP technology can provide the necessary security and privacy infrastructure for big data analytics. People can entrust their data in polymorphically encrypted form, and each time decide later to make (parts of) it available (decryptable) for specific parties, for specific analysis purposes. In this way users remain in control, and can monitor which of their data is used where by whom for which purposes.

The polymorphic encryption infrastructure can be supplemented with a pseudonymisation infrastructure which is also polymorphic, and guarantees that each individual will automatically have different pseudonyms at different parties and can only be de-pseudonymised by participants (like medical doctors) who know the original identity.

This white paper provides an introduction to Polymorphic Encryption and Pseudonymisation (PEP), at different levels of abstraction, focusing on health care as application area. It contains a general description of PEP, explaining the basic functionality for laymen, supplemented by a clarification of the legal framework provided by the upcoming General Data Protection Regulation (GDPR) of the European Union. The paper also contains a more advanced, mathematically oriented description of PEP, including the underlying cryptographic primitives, key and pseudonym managment, interaction protocols, *etc.* This second part is aimed at readers with a background in computer security and cryptography. The cryptographic basis for PEP is ElGamal public key encryption, which is well-known since the mid 1980s. It is the way in which this encryption is used — with re-randomisation, re-keying and re-shuffling — that is new.

The PEP framework is currently elaborated into an open design and open source (prototype) implementation at Radboud University in Nijmegen, The Netherlands. The technology will be used and tested in a real-life medical research project at the Radboud University Medical Center.

# Contents

# Chapter 1

# An introduction to PEP

This report is not a scientific research paper describing new deep mathematical results. Instead, it is meant to explain. It explains a novel approach to secure data management, called Polymorphic Encryption and Pseudonymisation (PEP). The underlying mathematical basis is surprisingly simple — for people with a reasonable background in cryptography — but at the same time surprisingly powerful. Its power lies in the new paradigm that it provides, and in the new applications that it enables. Hence the value of the work lies not so much in the depth of its cryptographic basis but in the breadth of the application scenarios. They may change the way we secure data in the era of big data analytics, with data coming from multiple sources.

A motivating aim for the development of PEP is to advance the security and privacy-friendliness of personalised medicine. This new trend in healthcare develops fine-grained personalised treatment methods based on statistical outcomes of large scale analysis of patient data. In personalised healthcare one has to deal with at the same time:

- identifyable medical data for the diagnosis and treatment of individual patients;
- pseudonymised patient data for large scale medical research;
- the need to ensure confidentiality, integrity, authenticity and availability of patient data;
- the ability to handle multiple sources of patient data, including in particular (wearable) self-measurement devices and apps.

The PEP framework is designed for this situation. It offers unprecented privacy-protection via encryption and pseudonymisation and at the same time it supports the basic data-access functionality for both treatment and research in personalised healthcare. Among the security goals listed in the third bullet, the PEP system concentrates on confidentiality. In a comprehensive approach, the other goals will have to be guaranteed via other means.

The PEP approach is applicable in many other areas than healthcare. However, this report concentrates on health informatics: it uses illustrations only from the healthcare sector and leaves it to the imagination of the reader to transfer the methodology to other sectors, for instance to handle sensor or surveillance data in the internet of things.

This report pays ample attention to explaining the new paradigm of 'polymorphic' encryption and pseudonymisation, together with its applications, especially in health care. The current first chapter is aimed at interested professinals: medical doctors, lawyers, managers, *etc.* It explains the relevant ideas via pictures, representing encryption of data as locking the data in a chest. The polymorphic character of our approach is described in terms of (blindly) tweaking (manipulating) of not only locks but also the content of these chests. The second chapter describes the underlying cryptographic ideas, which basically amount to clever use of the homomorphic properties of ElGamal encryption. That chapter is written for specialists in computer security and cryptography.

## 1.1 The rigidity of traditional encryption

Many people nowadays use self-measurement devices and apps for keeping track of their health and activities, for instance via watches that count steps, measure blood pressure, or even take electrocardiograms (ECGs). These devices and apps handle sensitive behavioural or medical data. Article 8 of the European Data Protection Directive (95/46/EC) qualifies health data as a special category of data to which a higher level of data protection applies. Processing of special categories of data is prohibited, unless an exception applies.

Many of these apps and devices transfer the measurements to some central database 'in the cloud' that is operated by (or on behalf of) the manufacturer. The data are then accessible for the user via special apps or web-based accounts. Duty of care applies. The transfer of data should only happen in encrypted form, as protection against eavesdropping. Once transferred, the data is ideally stored in encrypted form too, so that a possible security incident does not immediately lead to loss of (plain, unencrypted) data. The party that possesses the decryption keys will have access to the sensitive data. These keys are needed to give users access to their own data. Hence it is usually the manufacturer who possesses the keys, and has access to all user data.

In modern data science, or (big) data analytics, data is useful for many purposes. Such flexible use of the data is hindered by encryption. Indeed, traditional encryption is always 'for a particular party', namely for the party that has the decryption key. No-one else can decrypt. The decision who can decrypt has to be taken at the moment of encryption. In a multiple-use scenario, where data are encrypted, many parties must have the key. This undermines the protection level.

The main benefit of polymorphic encryption is that it allows more flexible usage scenarios, where the choice who is allowed to decrypt can be postponed, while retaining data protection. This will be explained pictorially in the next

section.

## 1.2 Traditional and polymorphic encryption, pictorially

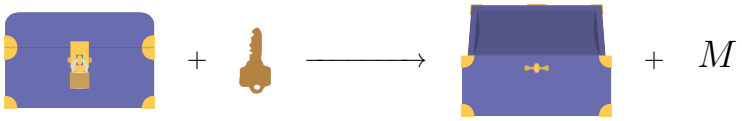
Encryption is a mathematical technique that makes data or messages unreadable, but in such a way that anyone in possession of a specific 'cryptographic' key can make the encrypted message readable again. We shall abstract from the method of encryption and describing it pictorially as putting a message in a chest with a lock. Only people with the appropriate key can open the lock and thus read the message.

Thus, **encryption** of a message $M$ can be described as:


(1.1)

The suggestion is that the message $M$ is now inside the chest, securely locked. The locked chest can remain in store, where it is, or it can be transported to another location.

The reverse process of **decryption** is depicted as unlocking the chest:




(1.2)

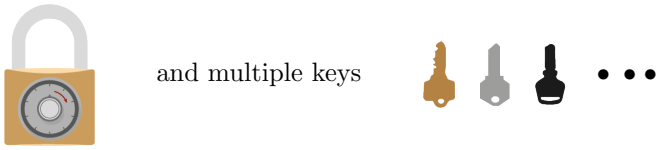
By opening the chest the original message $M$ pops out. This opening can be performed only by someone who possesses the single key that fits the lock. We consider the ideal situation where the right key is absolutely necessary, and the chest cannot be openend in any other way, for instance by force. It may be possible though that multiple people have a copy of the single key that opens the lock.

Via this chest metaphore we can explain some basic cryptographic terminology. The message $M$ is called the plaintext. In encrypted form, locked inside the chest, it is called the ciphertext. The open lock is called a public key, and the key that opens the lock is the associated private key.

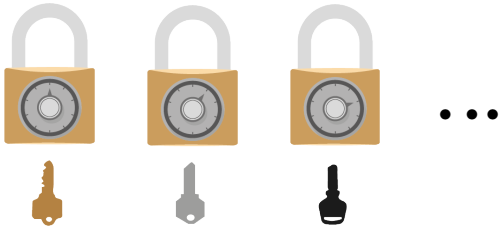
When I'm the only person that has such key, then I can distribute many open locks for this particular key publicly available, so that others can use it to encrypt message for me as in (1.1), which only I can decrypt, as in (1.2). This is the essence of what is called 'public key' encryption.

An important point is: once the lock is closed, there is only one key that can open it. This is the 'rigidity' that we discussed in Section 1.1: if multiple people need access, they all need a copy of the key. We would like to have more flexibility.

Next we consider a similar metaphore for *polymorpic* locks. We depict this new concept as a lock with a wheel:

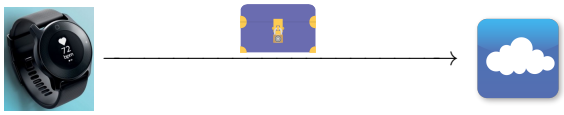



The wheel on the lock is not some additional protection measure, like a dial on a safe. Instead, by turning the wheel different keys fit and open the lock.



Notice the different positions of the wheel. If it is 'up', only the brown key on the left opens the lock. But if the wheel is moved one notch to the right, the grey key in the middle fits, and no longer the brown one. And if the wheel is moved another notch, the black key on the right fits exclusively. The technique allows an unlimited number of such notches — and thus an unlimited number of corresponding keys — for a single polymorpic lock. In principle, anyone can turn this wheel, but special knowledge is needed to select the right wheel position, out of the many possible ones, so that a particular key fits.

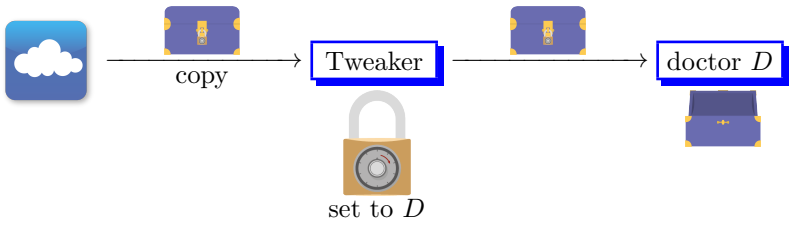
How such polymorphic locks can be realised mathematically will be described in Chapter 2. Here we illustrate how they can be used. We start with a simple scenario where sensitive data from some self-measurement device, like a watch, need to be stored securely in some cloud Storage Facility. We consider the situation where the data is put in a chest, with a polymorphic lock, as described above:

 (1.3)

The cloud provider cannot access the data from the watch, since they are encrypted in a polymorphic manner.

Next we consider the situation where a medical doctor, denoted abstractly as doctor $D$, needs to get access to the data from this device. This can be realised via an intermediate party called the Tweaker. It is a central converter who exclusively knows how to turn the wheel on a polymorphic lock so that

keys of specific parties fit. We describe the interaction as follows.




(1.4)

The process of turning the wheel on the lock will be called re-keying, see Lemma 2.1.2 (2) on page 25 for a mathematical description.

In these diagrams (1.3) and (1.4) we see the power of polymorphic encryption: data is stored in encrypted form, where the cloud storage provider cannot get access. Who does get access can be decided later, by suitably turning the wheel on the lock. In the illustration it is doctor $D$, but it can well be doctor $E$ at some other stage, or a medical researcher, or a service provider.

There is this (trusted) intermediate party, called the Tweaker, who knows how to turn locks in a specific manner, so that specific participants can open the lock. Thus the Tweaker has a crucial, powerful position. But the Tweaker works blindly: it cannot see the data (the contents of the chest); it can only turn the wheel on a lock, on the outside of the chest. (Here we implicitly assume that the Tweaker is not in possession of any of the possible keys.)

When a suitable authentication and access infrastructure is added, the user can set rules for the Tweaker and control usage of the data. The user can then make his/her own data available, for instance for (public) scientific research, but not for (private) commercial research. Or (s)he may control which members of the medical profession can(not) access which data. This will be elaborated in Section 1.4 below.

If this PEP approach develops into a standard, and 'PEP-compliant' wearables and apps become available, users can be in control of their data. The novel idea is that polymorphic encryption works in a generic manner, and the decisions about who can decrypt need not be taken at the time of encryption. The encrypted ciphertext can be tweaked later, in a blind manner, so that chosen participants can decrypt and get access to the data.

## 1.3 Polymorphic pseudonymisation

The PEP methodology consists of both encryption and pseudonymisation, in polymorphic form. This section explains pseudonymisation, also via pictures with chests, but with an extra wheel, on the chest itself.

First we have to look at identities and pseudonyms. We assume that each participant in the system has a unique (personal) identity, written as pid. This is typically a special number, like a social security number or some other (medical) registration number or identifier. We abstract away from the details: for participant $A$ we shall write $\mathrm{pid}_A$ for an identifier that is uniquely associated

with $A$. Such 'global identifiers' are useful for linking data accross different databases, but they form serious privacy risks — since they make it possible to break local contexts [16] — and also security risks — for instance in the form of identity fraud.

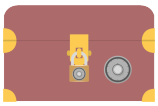
These pid's form the basis for 'local' pseudonyms. Each participant will have a different pseudonym at different parties. For instance, I will have different pseudonyms at doctors $X, Y, Z$, and at medical research groups $U, V, W$. The reason is as follows. These parties could somehow loose their data, or even maliciously combine data with others. If different parties use different pseudonyms for the same patient, it is in principle not possible to combine the data — at least not on the basis of identifiers. In general, one speaks of 'domain-specific' pseudonyms; they make it impossible to link identities accross different domains.

We shall write:

$$\mathrm{pid}_A@B \qquad \text{for} \qquad \text{the pseudonym of } A \text{ at } B.$$

Thus, the different pseudonyms of patient $A$ at doctors $X, Y, Z$ are written as $\mathrm{pid}_A@X, \mathrm{pid}_A@Y, \mathrm{pid}_A@Z$ respectively. They will sometimes be called 'local' pseudonyms, since they are local to these different doctors. Doctors will thus store both the real name/identity of their patients and their local pseudonyms. Researchers will only have (their own) local pseudonyms, but not identities of patients.

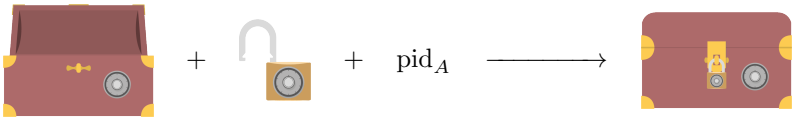
The Tweaker plays a central role in forming these local pseudonyms, in a blind manner. We represent this again via a chest with a (polymorphic) lock:



These new chests have a different color: we use these red chests for pseudonyms, and, like before, the blue chests for data.

But more importantly, these pseudonym chests have a wheel themselves: there is not only a wheel on the lock, to make it polymorphic, but also a wheel on the side of the chest, next to the position of the lock. By turning this second wheel, the contents of the chest can be changed, in a blind manner, without opening the chest. We use this as follows.

A **polymorphic pseudonym** for participant $A$ is formed by putting $A$'s personal identifier $\mathrm{pid}_A$ locked in a red chest with a polymorphic lock:




(1.5)

The two main points are:

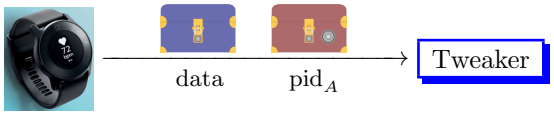
- a local pseudonym $\mathrm{pid}_A@B$ can now be constructed inside the chest by turning the wheel on the chest to position $B$; this process will be called re-shuffling, see Lemma 2.1.2 (3) later on;

- if the wheel on the lock is also put in position $B$, then $B$ can open the locked chest and find the local pseudonym $\text{pid}_A@B$.

The resulting box, with both wheels suitably turned, will be called an encrypted pseudonym.

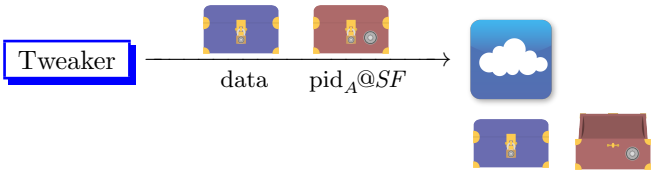
This set-up is extremely useful, as will be illustrated next. We return to the smart watch scenario from the previous section, but extend the protocol with the identity of the user.

Let $A$ be the user/owner of the smart watch. Assume that the watch somehow contains the identity $\text{pid}_A$ of the owner, in a chest (as polymorphic pseudonym). When the watch needs to off-load (senstive) data to a Storage Facility, it sends two chests to the Tweaker:




(1.6)

The Tweaker does not touch the first (blue) data chest. But it turns the two wheels on the second (red) identity chest, both to position $SF$, for the Storage Facility. As a result, the red chest contains the local pseudonym $\text{pid}_A@SF$ of user $A$ at the Storage Facility. The Tweaker then passes both chests on, for storage:



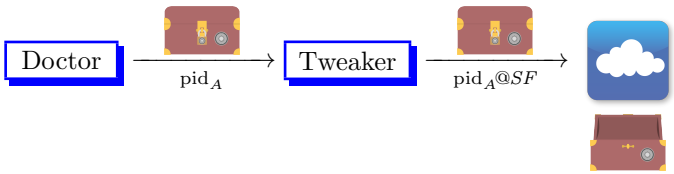

(1.7)

Because the wheel on the lock of the identity chest has also been turned to position $SF$, the Storage Facility can open this chest, so that the local pseudonym $\text{pid}_A@SF$ pops out. $SF$ uses this pseudonym as a database key, where the blue data chest is stored, see Figure 1.2 below for more information.

The same procedure is followed the next time that the watch needs to off-load data. The same pseudonym $\text{pid}_A@SF$ pops out on the $SF$ side, and the new blue chest is stored, under the same database key, next to the earlier blue chest. In fact, the same procedure is also followed when a medical doctor has examined $A$ and wants to store the diagnosis. The doctor — or, the computer of the doctor — puts the diagnosis data in a blue chest with a polymorphic lock, and the patient identifier $\text{pid}_A$ in a red chest and sends both of them off to the Tweaker, as in (1.6). The Tweaker then proceeds as in (1.7), so that the encrypted diagnosis data is added to the database record with the watch data, under the same database key. Figure 1.2 gives a sketch of such a record.

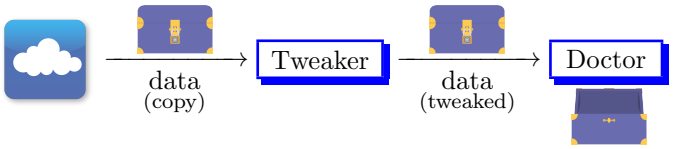
Next we look at a retrieval scenario. We assume that person $A$ visits a medical doctor $B$, who needs to retrieve some files about $A$ from the Storage Facility. At this stage we ignore the issue of file selection, and assume that it is somehow known how the appropriate file (in a chest) must be chosen. Doctor $B$

knows (or gets) the identifier $\text{pid}_A$ of $A$, and sends it off in a red identity chest to the Storage Facility, via the Tweaker:




(1.8)

As before, the Tweaker sets both wheels, on the chest and on the lock, to position $SF$, so that $SF$ can open the chest and find the local pseudonym $\text{pid}_A@SF$. The Storage Facility then looks up the requested data, in a blue data chest, and returns the locked chest via the Tweaker. The Tweaker changes the polymorphic lock so that the key of the doctor fits, like in (1.4):




(1.9)

To summarise, the PEP methodology provides:

1. storage of encrypted, pseudonymised data, so that an inquisitive, malicious, or poorly protected Storage Facility can do little to harm the confidentiality of the data;
2. combined storage of data stemming from the same person but via different sources/devices;
3. retrievablility of the data for a specific person, by an authorised doctor or by the person him/herself.

There is more functionality that we do not discuss in this informal description. An important one is 'pseudonymous data sharing', where medical researchers can get access — typically after approval of their research plan by some oversight committee — to pseudonymised but unencrypted data. It may happen that during medical research, a beneficial or alarming signal is found in the medical data of a particular person, say $A$. In case of such a 'coincidental finding' the pseudonym of $A$ at the research group can be translated back to the local pseudonym for a medical doctor of $A$, who can link the pseudonym to the real identity, and inform $A$. Thus, de-pseudonymisation can only happen by parties who already know the original identity, see Subsection 2.3.3 below for details.

An overview of the different parties and of the data flows between them is given in Figure 1.1.

## 1.4 Authentication, authorisation, and selection

The above informal description covers the core functionality of the PEP approach. In order to develop PEP into a practical system with appropriate

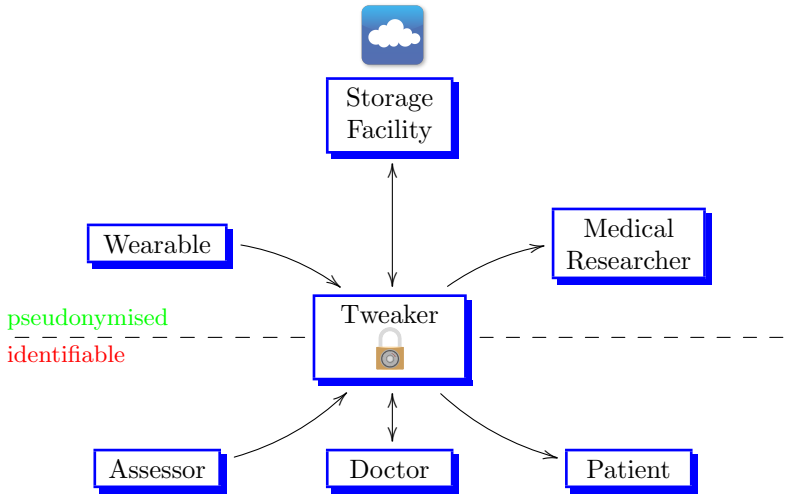



Figure 1.1: An overview of the interactions via the Tweaker, who can tweak locks and pseudonyms

guarantees more functionality needs to be added. In particular, an infrastructure must be added 'around' the Tweaker in Figure 1.1 for authentication and authorisation of the various parties involved, and also for logging. This section briefly discusses these matters at a conceptual level.

First we should say a bit more about data storage. So far we have mentioned only that local pseudonyms $\mathrm{pid}_A@SF$ at the Storage Facility are used as database keys. We have not said anything about the structure of database records. The medical content of each record will be encrypted, via the blue data chests used above, but some metadata needs to be added, so that doctors, researchers or others can access the appropriate data parts. A very simple picture of such a database entry is described in Figure 1.2, in order to convey the idea.

Thus, every blue chest that is stored, like in (1.6) and (1.7), must be accompagnied by appropriate metadata (labels, dates, sources), so that it can be placed appropriately in this table. Similarly, every retrieval request in (1.8) should involve a description of the specific data that is requested in terms of the entries in Figure 1.2.

The precise organisation of these database records is not relevant in the current context. Instead of the rather arbitrarily chosen labels in the left column in Figure 1.2, a standard medical classification system should be used. The meta-data could also be cryptographically protected, to make them invisible to the Storage Facility. We will not go into this matter here.

Instead, we turn to authentication. Consider the situation in Figure 1.1 where a patient $A$ wants to see the data that are stored about him/herself. This requires $A$ to prove that (s)he really is $A$, via a properly strong authen-

| $\text{pid}_A$@*SF* | **date** | **source** | **content** |
|---|---|---|---|
| **Identity insurance** | 1/6/2003 | doctor *X* |  |
| **ECG** | 20/3/2016 | watch |  |
| **Pulse** | 20/3/2016 | watch |  |
| **Radiology** | 15/2/2015 | UMCRadboud |  |
| ⋮ | ⋮ | ⋮ | ⋮ |

Figure 1.2: Sketch of a database record, with local pseudonym $\text{pid}_A$@*SF* as database key.

tication mechanism, so that $A$ can only see his/her own files. The (technical) details of this authentication mechanism are not relevant here. It should give an acceptable level of certainty that $\text{pid}_A$ really is $A$'s personal identifier, so that it can be used to retrieve data, like in protocols (1.8) and (1.9), where the final recepient is not 'Doctor', but $A$. This means that $A$ should have his/her own private key and client-side software to decrypt, that is, to open blue chests.

Additionally we foresee that, after authentication, user $A$ gets access to a 'dashboard' that gives an overview of, among other things:

- what data is stored about $A$, that is, a listing of the record $\text{pid}_A$@*SF*, as in Figure 1.2;
- log files, describing who has accessed which data of $A$ at which time;
- a configurable set of access rules, where user $A$ can decide which medical staff can get access to which data; these rules may for instance be based on white listing, on black listing, or on a combination;
- a similar set of rules for other use of the data, together with purpose descriptions. This 'other' usage may include, for instance, commercial or non-commercial medical research, or additonal services, based on a Data Licensing Agreement (DLA), see Section 1.5. In principle, the whole set-up also allows that users sell their data in pseudonymous form, but still get the revenues individually.

The precise organisation of such a dashboard involves many policy decisions that are outside the scope of this paper.

We have discussed authentication of patients. In a similar way, medical staff will have to authenticate itself, not only for access control but also for logging purposes, see Section 2.4 for further discussion.

Finally, before moving on to the cryptographic details, we like to emphasise the following points.

1. Pseudonymisation in the PEP framework only concentrates on cryptographic protection of identifiers. Possible de-pseudonymisation (or 're-identification') via the data is a totally different matter (see *e.g.* [7]), which is very important, but out of scope. Such de-pseudonymisation may happen simply because data contain identifiers — which happens for instance frequently with radiological images, where the names of the patients are embedded — or because combinations of data lead to a profile that fits only one or a few people. There are many studies — see *e.g.* [15] about the famous Netflix and AOL cases — showing that re-identification is often easier than expected, especially in combination with other databases or with public information — for instance from social media. This issue is highly relevant in a medical setting, see *e.g.* [9].

2. In the PEP framework the user is not in complete control over his/her data. For instance, a fraud-monitor entity may be added to the picture in Figure 1.1. If certain conditions determined by an anti-fraud policy are met, the Tweaker can be ordered to turn wheels on locks in such a way that the fraud-monitor can decrypt. Such a set-up can be understood as a backdoor into the encryption. It may be defendable, or even desirable, in some situations. In that case we advocate maximal transparancy and accountability.

## 1.5 Legal framework

The Polymorphic Encryption and Pseudonymisation (PEP) framework addresses the issue of an individual's control over his/her sensitive personal data. Art. 9 of the General Data Protection Regulation (GDPR) defines the following data as sensitive: 'personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade-union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation'. The processing of such data is prohibited by default; the main exception for this prohibition is explicit consent for one or more specific purposes (though other exceptions may be relevant in the context of health, see *e.g.* art. 9.2 (h) and 9.3 — mainly concerning processing of health data by medical practitioners under an obligation of confidentiality).

In the context of medical research patients are often confronted with so-called 'blanket' of 'broad' informed consent forms. When analysing such forms the consent is not really blanket but may indeed be overly broad. The purpose specified in such consent forms is clearly medical research in the context of a particular disease or medical field. If well explained this forms an appropriate purpose. The broadness resides in the inclusion of secondary use for compatible purposes regarding similar medical research, either in the course of a longitudinal study or for other studies. The latter easily turns the consent into an ambiguous consent if one is not aware of samples being used for entirely different types

of medical issues. More problematic is the inclusion of consent to share data with commercial partners who may even vest intellectual property rights in the results. In the case of PEP this should be excluded as each new entity processing the data will require its own access. In the final paragraph, when discussing Data Licensing Agreements we will return to this point.

When up and running, the PEP system will afford a service that enables a person to ensure that any data (s)he wishes to store and make available to other service providers is immediately encrypted and kept in store for future use. The data may have been submitted by the data subject or *e.g.* by his/her doctor or teacher, but it may also have been captured by applications that track behavioural data (as with health dedicated social networking sites, or quantified self applications). The PEP framework thus enables a service that is also provided by a number of other so-called personal data management systems, but on top of that, PEP provide two crucial additional functionalities: it affords (1) targeted reuse of sensitive data for big data analysis, in combination with, and based on, (2) the sharing of dedicated pseudonymous datasets. Both will be discussed below with reference to the upcoming General Data Protection Regulation (GDPR), to show how PEP will contribute to legal compliance and data protection by design. This will be followed by a discussion of three further points of relevance with regard to the GDPR, notably the special regime for pseudonymisation, the requirement of data protection by design and the idea of a modular data licensing agreement.

**Targeted reuse of sensitive data for big data analytics** First, the PEP framework enables a person to decide at any moment in time to share (parts of) his/her stored data with any specified and identified third party, based on a targeted consent that specifies for what purpose the data may be used (art. 6.1 (a) GDPR stipulates that consent can only be given for one or more unambiguously specified purposes). This consent concerns either historical data, streaming data or future data or any combination thereof and should preferably be part of a license to use the data for a specific period of time, after which they must be deleted (if consent is not renewed). To the extent that the consent is well-informed this provides a valid legal ground to process sensitive data (art. 8 (a) and 9.2 (a) GDPR), and to the extent that the purpose is explicitly specified data controllers will have complied with another core condition for fair and lawful processing of personal data (art. 5.1 (b) GDPR). Obviously, once the specified purpose is exhausted the data must be deleted. Processing for another, compatible purpose must be communicated to the data subject, and processing for another, incompatible purpose is prohibited. The latter will require a new consent and/or personal data licensing agreement, and a new decryption key.

The same data or parts thereof can be licensed to different third parties, each of whom will have an 'own', unique decryption key that is linked to specific data, a specified purpose and an identified data controller. To ensure the reliability and lawfulness of the system the consent as well as the license should prohibit any party from sharing the data with others. Any party wishing to process the

data will have to connect with the system to obtain its own key, specifying its own specific purpose and clarifying the time period for which it seeks permission.

By checking into the system, a data subject can see — via the dashboard from the previous section — which of her data she has licensed to what identified parties for what purposes. She can also check what consent she has withdrawn (art. 7.3 GDPR stipulates that 'consent can be withdrawn at any time').

**Sharing dedicated pseudonymous datasets: to have one's cake and eat it too** Second, the PEP framework entails that the data that are shared are pseudonymised, meaning that they cannot directly identify the data subject to which they relate. This reduces the risk of re-identification, thus contributing to data minimisation (art. 5.1 (c) GDPR stipulates that data processing must be 'adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed', which is qualified as 'data minimisation'). In fact, the PEP system constitutes a form of data protection by default (DPbD), thus ensuring compliance with the GDPR, which requires such DPbD in art. 25. The pseudonymisation thus enables big data analysis without access to the 'raw' data that contains sensitive personal information. Because each data controller has an 'own', unique key, data cannot easily be linked between different data controllers, further reducing the risk of re-idenfication, while affording the sharing of dedicated pseudonymous datasets (dedicated to processing for a specific purpose by an identified data controller).

The PEP system has the added advantage that, if analysis of the data generates specific risks in the data set with regard to an unidentified individual (*e.g.* health risks), the person who submitted the raw data can be notified, enabling re-identification. This will be either the data subject, or *e.g.* the doctor that submitted medical data in the first place.

In a sense, PEP enable us to have our cake and eat it too: we have privacy, but we also enable epidemiological research that would otherwise infringe our privacy, and under specified conditions re-identification is possible.

**The role of pseudonymisation in the GDPR** In recital 28 the GDPR states that '[t]he application of pseudonymisation to personal data can reduce the risks to the data subjects concerned and help controllers and processors to meet their data-protection obligations'. Art. 4.5 of the GDPR defines pseudonymisation as 'the processing of personal data in such a way that the data can no longer be attributed to a specific data subject without the use of additional information, as long as such additional information is kept separately and subject to technical and organisation measures to ensure non-attribution to an identified or identifiable person'. This definition clarifies that pseudonymous data is by definition personal data, meaning that the GDPR applies. Based on recital 26 we can conclude that encryption is a form of pseudonymisation, even if the data controller cannot access the additional data (identifier); as long as someone has a key de-identification is not irreversible and therewith the data are not considered anonymous. Pseudonymisation is, however, explicitly qualified

as 'data protection by default', which refers to architecting data minimisation into the relevant technical systems, and similarly qualified as what could be coined as an example of 'security by design' in art. 33.1 (a) GDPR. Obviously the extent to which pseudonymisation 'counts as' effective data protection will depend on the potential for its reversal. Recital 75 refers to this when summing up how data controllers should assess the risks of their processing operations for the rights and freedoms of data subjects: 'The risk to the rights and freedoms of natural persons, of varying likelihood and severity, may result from personal data processing which could lead to physical, material or non-material damage, in particular: where the processing may give rise to discrimination, identity theft or fraud, financial loss, damage to the reputation, loss of confidentiality of personal data protected by professional secrecy, *unauthorised reversal of pseudonymisation*, or any other significant economic or social disadvantage; where data subjects might be deprived of their rights and freedoms or prevented from exercising control over their personal data; where personal data are processed which reveal racial or ethnic origin, political opinions, religion or philosophical beliefs, trade-union membership, and the processing of genetic data, data concerning health or data concerning sex life or criminal convictions and offences or related security measures; where personal aspects are evaluated, in particular analysing or predicting aspects concerning performance at work, economic situation, health, personal preferences or interests, reliability or behaviour, location or movements, in order to create or use personal profiles; where personal data of vulnerable natural persons, in particular of children, are processed; or where processing involves a large amount of personal data and affects a large number of data subjects' (own emphasis).

The pseudonymisation that PEP enables will not always constitute pseudonymisation in the legal sense. This is due to the fact that in this case only the identifier is replace byd a pseudonym, while the data may enable identification due to its linkability with other data (within the same database or after fusing databases) or due to unique attributes that make possible the singling out of the individual (which may also relate to the size of the data base). This means that those gaining access to data via the PEP framework still have a duty of care to ensure the security of the data and — obviously — the legitimacy of its processing. The risks that processing these data pose to the rights and freedoms of data subjects are, however, substantially reduced by pseudonymisation, which will most probably count as a form of data protection by design (and contribute to compliance with the requirement of technical and organisation measure required in art. 24 and 32 of the GDPR).

**The role of Data Protection by Design and Default in the GDPR**
*Data Protection* by Design (DPbD) must not be confused with *Privacy* by Design (PbD), despite numerous links and overlaps. The core distinction is that whereas PbD may be an ethical requirement, DPbD will soon be a legal requirement. It is also important to note that privacy is a freedom right, making it very hard to define, let alone design or engineer. DPbD requires building data

protection into the technical and organizational architecture of personal data processing systems. It differs from *e.g.* privacy enhancing technologies (PETs) in that DPbD is not about add-ons but about requirements that should inform the construction of the system from its inception.

Art. 25.1 GDPR (data protection by default) states: 'Taking into account the state of the art, the cost of implementation and the nature, scope, context and purposes of processing as well as the risks of varying likelihood and severity for rights and freedoms of natural persons posed by the processing, the controller shall, both at the time of the determination of the means for processing and at the time of the processing itself, implement appropriate technical and organisational measures, *such as pseudonymisation*, which are designed to implement data-protection principles, such as data minimisation, in an effective manner and to integrate the necessary safeguards into the processing in order to meet the requirements of this Regulation and protect the rights of data subjects (my emphasis)'.

Art. 25.2 GDPR (data protection by design) states: 'The controller shall implement appropriate technical and organisational measures for ensuring that, by default, only personal data which are necessary for each specific purpose of the processing are processed. That obligation applies to the amount of personal data collected, the extent of their processing, the period of their storage and their accessibility. In particular, *such measures shall ensure that by default personal data are not made accessible without the individual's intervention to an indefinite number of natural persons* (our emphasis)'. The PEP approach arguably and demonstrably provide appropriate technical measures to contribute to compliance with both data protection by default and by design.

**Data Licensing Agreement (DLA)** The processing of personal data can be based on one of six legal grounds (art. 6.1 GDPR). Since many entities involved in big data analysis are not aware of this (thinking there is only consent), these six ground are summed up below.

a). 'the data subject has given *consent* to the processing of his or her personal data for one or more specific purposes;

b). processing is necessary for the performance of a *contract* to which the data subject is party or in order to take steps at the request of the data subject prior to entering into a contract;

c). processing is necessary for compliance with a *legal obligation* to which the controller is subject;

d). processing is necessary in order to protect the vital interests of the data subject or of another natural person;

e). processing is necessary for the performance of a task carried out in the *public interest* or in the *exercise of official authority* vested in the controller.

f). Processing is necessary for the purposes of the *legitimate interests pursued by the controller* or by a third party, except where such interests are overridden by the interests or fundamental rights and freedoms of the data subject which require protection of personal data, in particular where the data subject is a child.

Point (f) of the first subparagraph shall not apply to processing carried out by public authorities in the performance of their tasks' (own emphasis).

PEP can be based on the first ground, consent, which is usually combined with a privacy policy or terms of service. We believe that its aims are better achieved by the introduction of a (modular) 'data licensing agreement' (DLA) that makes sure that data are only processed insofar as necessary for the performance of the agreement by a party that is not allowed to share the data with other parties. The latter should always conclude their own DLA to obtain their own key. This ensures that data subjects have a clear overview of the parties that process their sensitive data.

The DLA can be short and comprehensive, containing a series of *general* clauses and a set of *modular* clauses part of which are optional, see under (A) and (B) below. To ensure that the data subject is aware of each stipulation it can best be accessed online such that each clause is shown on a separate screen. This gives people the option to quickly click through the entire DLA, but at the same time they are tempted to read each clause with attention. Various types of animation can be designed to make the content accessible and easy to grasp. The contract should be available behind a button on the platform of the PEP framework provider.

**A). General clauses** The DLA starts with identifying the parties to the contract: (1) the data subject: a patient or *e.g.* a user of a health App; and (2) the data controller(s): an identified health-App service provider, doctor, medical specialist or *e.g.* a hospital, insurance company, research institute or pharmaceutical company.

Next, the DLA will stipulate the obligations of the data *subject* and the data *controller(s)*. The data subject:

- licenses the identified data controller(s) who is (are) a party to the DLA to use (process):
  - a specified set (stream) of her or his personal data;
  - for explicitly specified purpose(s);
  - clearly expressing unambiguous and informed consent for the processing of his or her sensitive data for the explicitly specified purpose(s).

The data controller(s):

- will use (process) the data:
  - only for the specified purpose(s) and — if necessary — for purposes that are deemed compatible (no re-use out of context);

  - employing additional techniques of anonymisation and pseudonymisation (if the data enables re-identification because it is linked with other data, or if it is unique within the dataset);
  - deleting the data once the purpose is exhausted;
  - but always within a specified time period (which can be extended with a renewal of the DLA if the purpose has not yet been exhausted);
  - confirming that the data subject has the right to withdraw her consent at any time (which only regards future processing);
  - providing an easy way to withdraw consent;
  - providing an easy way to receive an electronic copy of the data processed;
  - explicitly confirming that it (they) will comply with all relevant data protection stipulations, notably those concerning the provision of information and taking note of the data subject's right to object against excessive or incorrect processing (which any data controller is legally obliged to do anyway);
- will not share the data with any third party, confirming it (they) will refer any third party to the PEP provider to obtain its (their) own key, specify its (their) own purpose(s) and conclude its (their) own DLA.

As indicated above, the latter should prevent all-or-nothing consent and provide patients with a fair and clear choice of whom to give access to their sensitive data. Indeed, without strict implementation of this clause most of the added value of PEP is lost.

**B). Modular clauses** The modular clauses may, for instance:

- specify the identity of the data processor(s), and/or
- specify whether data may be processed outside the EU (based on what legal safeguards), and/or
- stipulate with whom the abstract results (which are not personal data) may or may not be shared, notably whether or not these abstract results (such as profiles) may be shared with commercial companies, and/or
- specify the type of analytics that will be employed, and/or
- specify the potential consequences of applying the inferences back to the data subject or to others.

The modularity of the DLA will prevent overly broad consent forms. For instance, patients may stipulate that abstract results based on the processing of their data cannot be shared with commercial partners. Some may find this highly problematic, because commercial partners fund medical research. The

question is whether it is fair that data subjects — at this moment — have no voice whatsoever in how the results of the analytics are distributed and monetized.

In terms of private law, the DLA would be an obligatory agreement, involving freely given unambiguous consent on both sides. The DLA assumes that in the case of joint controllers these conclude a prior contract that binds each of them to the DLA.

The advantage of having a DLA instead of a dynamic consent or permission system is that the articulation and signing of the DLA creates awareness of the direct relationship between the data subject and the party that wishes to process his/her data as part of big data analytics. This prevents undesirable network effects of secondary use by unidentified parties. It will also give data subjects easy access to an overview of who can lawfully process which of her pseudonymous data how for what purposes and for how long[1].

[1]For further reading on the use of DLAs see `www.usemp-project.eu`.

# Chapter 2

# The cryptographic basis of PEP

The description given below is definitely more technical than in the previous chapter, but it is not an implementation manual for PEP. Our focus is still on explaining the ideas and techniques. We occasionally allow ourselves to deviate from what will or should be done in practice, to order to avoid unnecessary complications that distracts from the essentials. In those cases we typically insert a 'warning' paragraph.

A naive form of local pseudonymisation is described in [20], via a combined hash of a name and a domain: a local pseudonym of $A$ at $B$ is obtained via a hash $\mathcal{H}(\text{pid}_A, B)$. This hash value is stored at a central party, together with the personal identifier $\text{pid}_A$, for de-pseudonymisation. The current approach is far more advanced. It is based on [21], see also [12]. Our aims are very much in line with [13] but our realisation provides better privacy protection, for instance through domain-specific (local) pseudonyms. However, we do not include the verification features from [4], which provide guarantees that the various participants include the right identity information into the various encryptions.

This chapter starts by recalling the basics of ElGamal encryption and by fixing notation. It is shown how three basic functions can be applied to ElGamal ciphertexts, for re-randomisation, for re-keying, and for re-shuffling. How these operations are applied is explained in Section 2.2, for the basic functionality of storing and retrieving data. Protocols for additional functionality is described in Section 2.3. Next, Section 2.4 proposes how to organise authentication, authorisation and logging in the PEP framework. Once this basic machinery is in place, we discuss two possible cryptographic enhancements in Section 2.5.

## 2.1 ElGamal revisited

The expression 'ElGamal' is used for one of the first asymmetric, public key crypto algorithms, named after its inventor [8]. It can be used both for en-

cryption and for digital signatures. Here we only use the encryption version. This section recalls the basic definitions and results, assuming familiarity only with elementary group theory. In particular, it describes three operations on ElGamal ciphertexts that form the basis for PEP.

ElGamal works in a cyclic group. Its security depends on the discrete logarithm (DL) problem in the group. In practice we shall use (subgroups of) elliptic curves [19] as groups, and so we prefer additive notation for a group $\mathbb{G} = (\mathbb{G}, +, 0)$. A possible instance of $\mathbb{G}$ is the Edwards Elliptic Curve denoted by Curve25519[1], which offers 128 bits of security, see Example 2.1.1 below for some more background information.

Translated to additive notation, the DL problem says:

> Let $g \in \mathbb{G}$ be a fixed group element; given $n \cdot g \in \mathbb{G}$, for some unknown number $n \in \mathbb{N}$, it is in general computationally infeasible to find $n$.

We use the dot notation $n \cdot g$, or sometimes simply $ng$, for the $n$-fold sum $g + \cdots + g \in \mathbb{G}$. The following basic equations hold.

$$\begin{array}{lll} n \cdot 0 = 0 & 0 \cdot g = 0 & (n+m) \cdot g = n \cdot g + m \cdot g \\ n \cdot (g + g') = n \cdot g + n \cdot g' & 1 \cdot g = g & (n \cdot m) \cdot g = n \cdot (m \cdot g). \end{array}$$

We recall the basics of ElGamal encryption. In this case we assume that $\mathbb{G}$ is a group and that $g \in \mathbb{G}$ is a fixed generator, or basepoint, for a subgroup, of (non-zero) order $N \in \mathbb{N}$. This means that $N$ is the least number with $N \cdot g = 0$. We write $|g| = N$ for the order of $g$, and typically assume that $N$ is a prime number. One writes $\langle g \rangle \subseteq \mathbb{G}$ for the subgroup of order $N$ generated by $g$, with elements of the form $n \cdot g$, for $0 \leq n < N$. In some cases $\mathbb{G} = \langle g \rangle$, but this is not necessary in what follows. If the order of $g$ is a prime number $p$, then the generated subgroup $\langle g \rangle$ is isomorphic to the field $\mathbb{F}_p$ of numbers $\{0, 1, \ldots, p-1\}$ below the prime $p$, via the mapping $\mathbb{F}_p \to \langle g \rangle$ given by $i \mapsto i \cdot g$. In fact, scalar multiplication $i \cdot g$ makes $\langle g \rangle$ a vector space over the field $\mathbb{F}_p$.

**Example 2.1.1.** As intermezzo, we briefly illustrate what these groups can look like in practice, for the earliermentioned group $\mathbb{G} = \text{Curve25519}$. This elliptic curve is a subset of points in the vector space $\mathbb{F}_p \times \mathbb{F}_p$ over $\mathbb{F}_p$, for the prime number:

$$\begin{aligned} p &= 2^{255} - 19 \qquad \text{(whence the name Curve25519)} \\ &= 57896044618658097711785492504343953926634992332820282019728792003956564819949. \end{aligned}$$

The curve itself is given by the equation $-x^2 + y^2 = 1 + dx^2y^2$, for the number:

$$\begin{aligned} d &= -\tfrac{121665}{121666} \mod p \\ &= 37095705934669439343138083508754565189542113879843219016388785533085940283555 \end{aligned}$$

[1] See `https://cr.yp.to/ecdh.html` for more information.

The default generator $g$ of a subgroup of Curve25519 is the point on the curve with coordinates:

$$\big(\,15112221349535400772501151409588531511454012693041857206046113283949847762202,$$
$$46316835694926478169428394003475163141307993866256225615783033603165251855960\,\big)$$

The generated subgroup $\langle g \rangle$ has cofactor 4, which means that it contains a quarter of the points on the curve.

We continue to describe the essentials of ElGamal encryption.

**Private key** The private key of a user is a natural number below the order of $g \in \mathbb{G}$. It is typically written as $x$, with $x < p = |g|$. Hence $x \in \mathbb{F}_p$.

**Public key** The public key $y \in \mathbb{G}$ associated with private key $x \in \mathbb{F}_p$ is the group element $y = x \cdot g \in \langle g \rangle \subseteq \mathbb{G}$. Due to the DL problem, $x$ cannot (feasibly) be obtained from $y$.

**Encryption** Let $M \in \mathbb{G}$ be the message that we wish to encrypt, with public key $y \in \langle g \rangle$. ElGamal encryption is 'randomised' or 'probabilistic': it uses randomness in each encryption so that encrypting the same message twice gives different ciphertexts. We choose a random number $r \in \mathbb{N}$, below the order of $g$, and use as ElGamal encryption of $M$ the pair of group elements:

$$\langle\, r \cdot g,\ r \cdot y + M \,\rangle. \tag{2.1}$$

We recall that a fresh (new) random number $r$ should be used for each encryption.

**Decryption** Let a ciphertext pair $\langle b, c \rangle \in \mathbb{G} \times \mathbb{G}$ be given. Let $x \in \mathbb{F}_p$ be the private key, corresponding to the public key $y = x \cdot g$ that has been used for the encryption. The ElGamal decryption of $\langle b, c \rangle$ is the group element:

$$c - x \cdot b. \tag{2.2}$$

(We use the letters $b$ for blinding and $c$ for cipher.)

**Correctness** First encrypting, then decrypting returns the original: if we start from plaintext $M \in \mathbb{G}$, and encrypt it with public key $y = x \cdot g$ and random $r$, giving ciphertext $\langle b, c \rangle = \langle r \cdot g, r \cdot y + M \rangle$, then decryption with private key $x$ yields the original message $M$, since:

$$c - x \cdot b = r \cdot y + M - x \cdot r \cdot g = r \cdot x \cdot g + M - x \cdot r \cdot g = M.$$

**Notation** We shall write $\mathcal{EG}$ for the ElGamal encryption function, but with a minor twist. We define:

$$\mathcal{EG}(r, M, y) = \langle\, r \cdot g,\ r \cdot y + M,\ y \,\rangle. \tag{2.3}$$

As before $r$ is the random number that needs to be different each time, $g \in \mathbb{G}$ is the generator of the underlying cyclic group $\langle g \rangle$, $M$ is the plaintext message that we wish to encrypt, and $y \in \langle g \rangle$ is the public key that is used in the encryption.

Notice that the function $\mathcal{EG}$ produces a 3-tuple in (2.3), instead of a 2-tuple in (2.1): its type is $\mathcal{EG} \colon \mathbb{F}_p \times \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G} \times \mathbb{G} \times \mathbb{G}$. This is purely for burocratic reasons: it makes it easier to formulate the results in Lemma 2.1.2 below.

Decryption still works essentially as in (2.2): given a ciphertext $\langle b, c, y \rangle = \mathcal{EG}(r, M, y)$ we decrypt it to $c - x \cdot g$. We do not use a special function or notation for decryption.

We now describe the three homomorphic properties of ElGamal that form the basis of PEP. They are used in the operations of *re-randomising*, *re-keying*, and *re-shuffling* that act on ciphertexts. The proofs of the relevant properties are simple calculations, but are included for convenience.

**Lemma 2.1.2.** *Let $g \in \mathbb{G}$ be a group element whose order $p = |g|$ is a prime number. We shall define three functions $\mathcal{RR}, \mathcal{RK}, \mathcal{RS}$ each with type:*

$$\mathbb{G}^3 \times \mathbb{F}_p \longrightarrow \mathbb{G}^3$$

*and describe their properties.*

1. *The* ***re-randomisation*** *of a triple $\langle b, c, y \rangle \in \mathbb{G}^3$ with random number $s < p = |g|$ is defined via the function:*

$$\mathcal{RR}(\langle b, c, y \rangle, s) \stackrel{def}{=} \langle\, s \cdot g + b,\ s \cdot y + c,\ y \,\rangle. \tag{2.4}$$

   *If the input $\langle b, c, y \rangle$ is an ElGamal ciphertext, then so is the output:*

$$\mathcal{RR}\big(\mathcal{EG}(r, M, y), s\big) = \mathcal{EG}(s + r, M, y). \tag{2.5}$$

   *Hence this output decrypts to the original message $M$ via the original private key $x$ (for which $y = x \cdot g$): the only effect of re-randomisation is to change the appearance of the ciphertext.*

2. *The* ***re-keying*** *with non-zero number $k < |g|$ is defined via the function:*

$$\mathcal{RK}(\langle b, c, y \rangle, k) \stackrel{def}{=} \langle\, \tfrac{1}{k} \cdot b,\ c,\ k \cdot y \,\rangle, \tag{2.6}$$

   *where $\frac{1}{k}$ is the multiplicative inverse of $k$ in the field $\mathbb{F}_p$. We then have:*

$$\mathcal{RK}\big(\mathcal{EG}(r, M, y), k\big) = \mathcal{EG}(\tfrac{r}{k}, M, k \cdot y). \tag{2.7}$$

   *This ciphertext decrypts to the orignal message $M$ via a different private key $k \cdot x$.*

3. *The* ***re-shuffling*** *with number $n \in \mathbb{F}_p$ is defined as a function:*

$$\mathcal{RS}(\langle b, c, y\rangle, n) \stackrel{def}{=} \langle\, n \cdot b,\ n \cdot c,\ y\,\rangle. \tag{2.8}$$

*Then:*

$$\mathcal{RS}\big(\mathcal{EG}(r, M, y), n\big) \;=\; \mathcal{EG}(n \cdot r, n \cdot M, y). \tag{2.9}$$

*Hence in this case we can decrypt with the original private key to a re-shuffled message $n \cdot M$.*

**Proof** All results (2.5), (2.7) and (2.9) are obtained by easy calculations.

1. We prove that equation (2.5) holds: re-randomisation (2.4) on an ElGamal encryption yields a new ElGamal encryption of the same message with the same public key, but with random $s + r$, since:

$$\begin{aligned}
\mathcal{RR}\big(\mathcal{EG}(r, M, y), s\big) &\stackrel{(2.1)}{=} \mathcal{RR}(\langle r \cdot g, r \cdot y + M, y\rangle, s) \\
&\stackrel{(2.4)}{=} \langle s \cdot g + r \cdot g, s \cdot y + r \cdot y + M, y\rangle \\
&= \langle (s + r) \cdot g, (s + r) \cdot y + M, y\rangle \\
&= \mathcal{EG}(s + r, M, y).
\end{aligned}$$

2. Re-keying of an encryption yields an encryption with a different key:

$$\begin{aligned}
\mathcal{RK}\big(\mathcal{EG}(r, M, y), k\big) &\stackrel{(2.1)}{=} \mathcal{RK}(\langle r \cdot g, r \cdot y + M, y\rangle, k) \\
&\stackrel{(2.6)}{=} \langle \tfrac{1}{k} \cdot (r \cdot g), r \cdot y + M, k \cdot y\rangle \\
&= \langle \tfrac{r}{k} \cdot g, \tfrac{k}{k} \cdot (r \cdot y) + M, k \cdot y\rangle \\
&= \langle \tfrac{r}{k} \cdot g, \tfrac{r}{k} \cdot (k \cdot y) + M, k \cdot y\rangle \\
&= \mathcal{EG}(\tfrac{r}{k}, M, k \cdot y)
\end{aligned}$$

   The adapted private key $k \cdot x \in \mathbb{F}_p$ has as associated public key $(k \cdot x) \cdot g = k \cdot (x \cdot g) = k \cdot y$, so it can be used to decrypt the manipulated ciphertext (2.6), giving the original message $M$.

3. We now have:

$$\begin{aligned}
\mathcal{RS}\big(\mathcal{EG}(r, M, y), n\big) &\stackrel{(2.1)}{=} \mathcal{RS}(\langle r \cdot g, r \cdot y + M, y\rangle, n) \\
&\stackrel{(2.8)}{=} \langle n \cdot (r \cdot g), n \cdot (r \cdot y + M), y\rangle \\
&= \langle (n \cdot r) \cdot g, (n \cdot r) \cdot y + n \cdot M, y\rangle \\
&= \mathcal{EG}(n \cdot r, n \cdot M, y). \qquad \square
\end{aligned}$$

Sometimes we shall combine the re-keying and re-suffling operations. The next result tells that the order of such combinations does not matter.

**Lemma 2.1.3.** *The re-keying and re-shuffling operations $\mathcal{RK}$ and $\mathcal{RS}$ from Lemma 2.1.2 commute. Explicitly:*

$$\mathcal{RS}\big(\mathcal{RK}(\langle b, c, y\rangle, k), n\big) \;=\; \mathcal{RK}\big(\mathcal{RS}(\langle b, c, y\rangle, n), k\big).$$

**Proof** This follows from an easy calculation:

$$\begin{aligned}
\mathcal{RS}\big(\mathcal{RK}(\langle b, c, y\rangle, k), n\big) &= \mathcal{RS}\big(\langle \tfrac{1}{k} \cdot b, c, k \cdot y\rangle, n\big) \\
&= \langle n \cdot (\tfrac{1}{k} \cdot b), n \cdot c, k \cdot y\rangle \\
&= \langle \tfrac{1}{k} \cdot (n \cdot b), n \cdot c, k \cdot y\rangle \\
&= \mathcal{RK}\big(\langle n \cdot b, n \cdot c, y\rangle, k\big) \\
&= \mathcal{RK}\big(\mathcal{RS}(\langle b, c, y\rangle, n), k\big). \qquad \square
\end{aligned}$$

Based on the above lemma we can combine re-keying and re-shuffling into a single function $\mathcal{RKS}\colon \mathbb{G}^3 \times \mathbb{F}_p^2 \to \mathbb{G}^3$ by:

$$\mathcal{RKS}(\langle b, c, y\rangle, k, n) \;=\; \langle \tfrac{n}{k} \cdot b, n \cdot c, k \cdot y\rangle. \tag{2.10}$$

However, for the time being we shall use these functions $\mathcal{RK}$ and $\mathcal{RS}$ for re-keying and re-shuffling separately. We return to this combination in Subsection 2.5.2.

Re-randomisation does not commute with re-keying, and also not with re-shuffling. But repeated re-randomisations can be combined, as in:

$$\mathcal{RR}(\mathcal{RR}(\langle b, c, y\rangle, s), s') \;=\; \mathcal{RR}(\langle b, c, y\rangle, s' + s).$$

Since $\mathcal{RR}(\langle b, c, y\rangle, 0) = \langle b, c, y\rangle$ we see that re-randomisation forms an action of the group $\mathbb{F}_p$ on $\mathbb{G}^3$.

## 2.2 Storing and retrieving data

This section illustrates how the three ciphertext manipulations $\mathcal{RR}, \mathcal{RK}$ and $\mathcal{RS}$ from Lemma 2.1.2 can be used to realise the basic PEP functionality of storing and retrieving data. The illustrations form a technical elaboration of the 'picture' examples from Sections 1.2 and 1.3. This section first explains in some detail the protocols for storing and retrieving data. The next section elaborates several other protocols.

Throughout we shall assume that there is a secret master private key $x$, with associated master public key $y = x \cdot g$, for a fixed group element $g$. This master key $x$ is securely stored by a trusted Key Server, in secure hardware, but it is never used for decryption, see Subsection 2.3.1. An overview of notation and terminology is given in Figure 2.1

**Warning 2.2.1.** *We shall use the above key pair $(x, y)$ for encryption of both data and identities, that is, in the terminology of Chapter 1, both for blue and for red chests. In practice we should use a separate key pair for each of these,*

*in order to exclude unintended mixing of encryptions. However, having distinct key pairs only clutters up the description and does not contribute much. It is important to be aware of this simplification.*

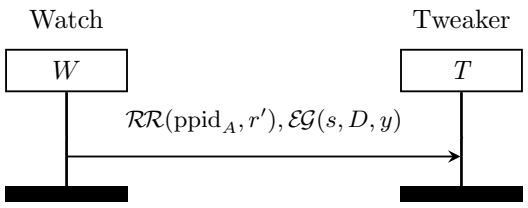
The protocol examples that we describe below are only meant to illustrate the re-randomisation, re-keying and re-shuffling operations. Many aspects are still missing that are important in actual use — notably authentication, see Section 1.4.

**Example 2.2.2.** We shall consider some elements from the scenario where a smart watch $W$ of a certain user $A$ periodically stores its behavioural/medical data in the Storage Facility. At first we only look at the interaction between the watch and the Tweaker $T$. We assume that the user/owner $A$ of the watch has a personal identity $\mathrm{pid}_A \in \mathbb{G}$ which is somehow embedded into the watch as a 'polymorphic pseudonym':

$$\mathrm{ppid}_A = \mathcal{EG}(r, \mathrm{pid}_A, y).$$

As before, $r$ is a random number, and $y$ is the master public key that is known to all parties.

Each time that the Watch needs to store some encrypted data $D$ externally, it will send a message to the Storage Facility via the Tweaker. The first step is described via the following message sequence chart. It corresponds to the earlier picture (1.6).




The first part $\mathcal{RR}(\mathrm{ppid}_A, r')$ of this message is a re-randomised version of the polymorphic pseudonym $\mathrm{ppid}_A = \mathcal{EG}(r, \mathrm{pid}, y)$, with an additional random number $r'$. By Lemma 2.1.2 (1) we have $\mathcal{RR}(\mathrm{ppid}_A, r') = \mathcal{EG}(r' + r, \mathrm{pid}_A, y)$. In this way the watch presents the same (encrypted) identifier each time in a different form, so that the Tweaker cannot link multiple messages from the same watch.

The second part $\mathcal{EG}(s, D, y)$ of the message is an ElGamal encryption of the data $D$ from the watch. Notice that the master public key $y$ is used for this 'polymorphic' encryption — with $s$ a random number. As we shall see below, this polymorphically encrypted message $\mathcal{EG}(s, D, y)$ can be tweaked at some later stage, via re-keying, so that it can be decrypted by a chosen participant.

**Warning 2.2.3.** *In practice it makes sense to do this data encryption $\mathcal{EG}(s, D, y)$ slightly differently. It is computationally more efficient, certainly for large data*

| **name** | **notation** | **description** | **known by** |
|---|---|---|---|
| master private key | $x$ | element of $\mathbb{F}_p$ | Key Server |
| master public key | $y$ | $y = x \cdot g$ | everyone |
| key factor for participant $A$ | $K_A$ | element of $\mathbb{F}_p$ | Tweaker |
| private key of participant $A$ | $x_A$ | $x_A = K_A \cdot x$ | $A$ |
| public key of participant $A$ | $y_A$ | $y_A = x_A \cdot g = K_A \cdot y$ | everyone |
| identity of participant $A$ | $\mathrm{pid}_A$ | element of $\mathbb{G}$ | $A$, medical staff |
| pseudonym factor for participant $A$ | $S_A$ | element of $\mathbb{F}_p$ | Tweaker |
| pseudonym of $A$ at participant $B$ | $\mathrm{pid}_A@B$ | $\mathrm{pid}_A@B = S_B \cdot \mathrm{pid}_A$ | $B$ |
| polymorphic pseudonym of $A$ | $\mathrm{ppid}_A$ | $\mathcal{EG}(r, \mathrm{pid}_A, y)$ | $A$, medical staff |
| encrypted pseudonym of $A$ at participant $B$ | $\mathrm{epid}_A@B$ | $\mathrm{epid}_A@B = \mathcal{EG}(r, \mathrm{pid}_A@B, y_B)$ | $B$ |

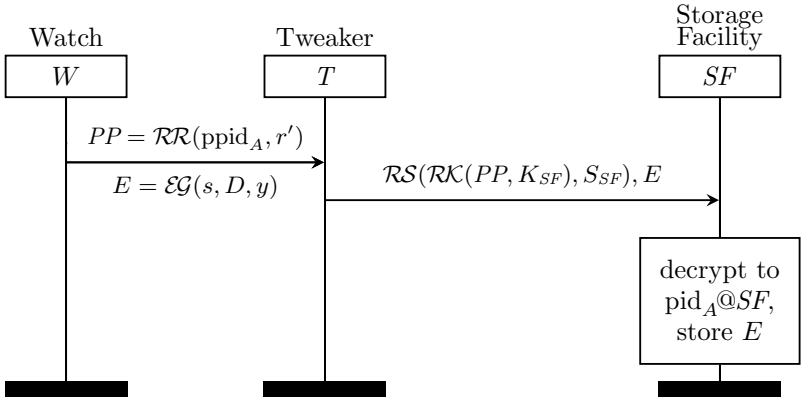
Figure 2.1: Names and notation, assuming a group element $g \in \mathbb{G}$ of prime order $p = |g| \in \mathbb{N}$

*blocks $D$, to use ElGamal encryption $\mathcal{EG}(s, K, y)$ only for a symmetric session key $K$, and use this $K$ to encrypt the actual data $D$. This is convenient because asymmetric (public key) encryption is generally much slower than symmetric encryption. However, to keep things simple, we shall write ElGamal encryption for data too.*

*Here we see that our polymorphic encryption is actually polymorphic* asymmetric *encryption. It would be useful to als have polymorphic* symmetric *encryption.*

**Example 2.2.4.** We continue the story from Example 2.2.2 for storing data from a watch and look at the subsequent interaction between the Tweaker and the Storage Facility (*SF*). The encrypted data $\mathcal{EG}(s, D, y)$, abbreviated as $E$ below, is simply passed on by the Tweaker. The Tweaker also passes on the encrypted identity of the user $A$, but only after manipulating it so that *SF* can decrypt the result to its local pseudonym $\mathrm{pid}_A@SF$ of the user. We first present

the earlier pictorial descriptions (1.6) and (1.7) as a chart, and then explain mathematically what is going on.




The numbers $K_{SF}$ and $S_{SF}$ are the key-factor and the pseudonym factor for the Storage Facility, see Figure 2.1 for an overview. These numbers are secret, and only known to the Tweaker.

- The *key-factor* $K_{SF}$ is generated when the *SF* joins the system, see Subsection 2.3.1 for details. This factor is used to compute the private key $x_{SF} \in \mathbb{F}_p$ of the storage facility as:

$$x_{SF} = K_{SF} \cdot x \mod p. \tag{2.11}$$

  The associated public key in $\mathbb{G}$ is:

$$y_{SF} = x_{SF} \cdot g = K_{SF} \cdot x \cdot g = K_{SF} \cdot y.$$

  The DL assumption guarantees that $K_{SF}$ cannot be computed from both $y$ and $y_{SF}$. The Tweaker knows $K_{SF}$ but cannot compute the private key $x_{SF}$ since it does not know the master private key $x$.

- The *pseudonym-factor* $S_{SF}$ for the Storage Facility is also generated and distributed when *SF* joins. It determines the pseudonym $\text{pid}_A@SF$ of user $A$ with identity $\text{pid}_A$ at the Storage Facility as:

$$\text{pid}_A@SF = S_{SF} \cdot \text{pid}_A \in \mathbb{G}. \tag{2.12}$$

  In general, the local pseudonym $\text{pid}_A@B$ of $A$ at $B$ is defined[2] as $S_B \cdot \text{pid}_A$, see Figure 2.1 for an overview.

[2]Instead of using $S_B \cdot \text{pid}_A$ as local pseudonym one can also use for instance $S_B \cdot h(\text{pid}_A)$, where $h$ is some (keyed) has function. In this way one can prevent that the Tweaker, if it ever sees a local pseudonym at $B$, can compute the actual identity $\text{pid}_B$, via division by $S_B$.

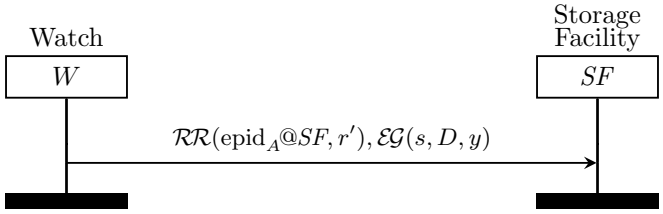
With these numbers $K_{SF}$ and $S_{SF}$ explained, we can now understand what the tweaked message in the above protocol contains:

$$\begin{aligned}
\mathcal{RS}(\mathcal{RK}(PP, K_{SF}), S_{SF}) &= \mathcal{RS}\Big(\mathcal{RK}\big(\mathcal{EG}(r'+r, \mathrm{pid}_A, y), K_{SF}\big), S_{SF}\Big) \\
&\overset{(2.7)}{=} \mathcal{RS}\Big(\mathcal{EG}(\tfrac{1}{K_{SF}} \cdot (r'+r), \mathrm{pid}_A, K_{SF} \cdot y), S_{SF}\Big) \\
&\overset{(2.9)}{=} \mathcal{EG}(\tfrac{S_{SF}}{K_{SF}} \cdot (r'+r), S_{SF} \cdot \mathrm{pid}_A, K_{SF} \cdot y) \\
&= \mathcal{EG}(\tfrac{S_{SF}}{K_{SF}} \cdot (r'+r), \mathrm{pid}_A@SF, y_{SF}) \\
&= \mathrm{epid}_A@SF.
\end{aligned}$$

The one-but-last equation holds by (2.11) and (2.12). We see that the recipient of this message, the Storage Facility, obtains an encrypted pseudonym $\mathrm{epid}_A@SF$ of $A$. It can decrypt the message, with its private key $x_{SF}$, and obtain the local pseudonym $\mathrm{pid}_A@SF$ for user $A$ with identity $\mathrm{pid}_A$. The Storage Facility will store the encrypted data $E$ in a database record, using this pseudonym $\mathrm{pid}_A@SF$ as database key, as sketched in Figure 1.2.

The *SF* learns nothing about the data, nor about its origin. A next batch of data from the watch will arrive at *SF* with the same pseudonym, so that *SF* can store it together with previous data, in the same database record. Even if new data arrives from a different source, say from a medical doctor who has examined user $A$, the resulting (encrypted) data will be added under the same pseudonym/database key $\mathrm{pid}_A@SF$ to the same record, as long as the doctor sends it in with the appropriate (polymorphically encrypted) identifier $\mathrm{pid}_A$.

In the end one may ask: why does the watch need to communicate via the Tweaker? It could send the data directly to Storage Facility, if it contains the encrypted pseudonym $EP = \mathcal{EG}(r, \mathrm{pid}_A@SF, y_{SF})$. The storage protocol can then be simplified to:




The first part $\mathcal{RR}(\mathrm{epid}_A@SF, r')$, for a fresh random number $r'$, is equal to $\mathcal{EG}(r'+r, \mathrm{pid}_A@SF, y_{SF})$. It can be decrypted by *SF*, giving the local pseudonym $\mathrm{pid}_A@SF$ that is used as database key for the record in which the encrypted data $\mathcal{EG}(s, D, y)$ should be placed.

In our overview picture in Figure 1.1 all communication goes via the Tweaker. This makes it possible to integrate authentication and logging with the activities of the Tweaker — as will be described in Section 2.4. The main disadvantage

of the above direct storage protocol is that it circumvents such logging and authentication. This may be acceptable for some of the relatively innocent data from a watch, but not for more senstive data from, for instance, an MRI scan.

Let's step back and see how and why this works. The main trick is to use 'diversified' private keys, which are derived from the master private key $x$, as $x_A = K_A \cdot x$, for participant $A$. The Tweaker knows these key factors $K_A$, and can thus re-key messages encrypted with the public master key $y$, without learning the content. Again, the Tweaker does not know $x$, and hence it also does not know the secret keys $x_A = K_A \cdot x$ of participants $A$. This corresponds to turning the wheel on a blue box from Section 1.2.

Similarly, for each participant $B$ there is a pseudonym factor $S_B$ such that the local pseudonym-of-$A$-at-$B$, written as $\mathrm{pid}_A@B$, is derived from the identity $\mathrm{pid}_A \in \mathbb{G}$ as $\mathrm{pid}_A@B = S_B \cdot \mathrm{pid}_A \in \mathbb{G}$. In this way each participant — including the Storage Facility, as we have seen in Example 2.2.4 — has its own pseudonym for each user. Figure 2.1 gives an overview. This local pseudonym is created for $B$ in a blind way, by re-shuffling and re-keying, that is, by turning both wheel on a red box and on its lock, like in Section 1.3.

We single out the most important operation, which we call PP2EP conversion.

**Definition 2.2.5.** *Let participant $A$ have personal identifier $pid_A$ and associated polymorphic pseudonym $ppid_A = \mathcal{EG}(r, pid_A, y)$. What we call the* PP2EP *conversion for another participant $B$ is the re-keying and re-shuffle application:*

$$\mathcal{RK}\big(\mathcal{RS}(ppid_A, S_B), K_B\big).$$

*This PP2EP conversion can be performed by the Tweaker (who knows $S_B, K_B$) and produces the encrypted pseudonym $epid_A@B = \mathcal{EG}(\frac{S_B}{K_B} \cdot r, pid_A@B, y_B)$ which can be decrypted by $B$, to its local pseudonym $pid_A@B$ of $A$.*

We conclude this section with one more scenario.

**Example 2.2.6.** Consider a doctor $B$ who wants to retrieve some data from the Storage Facility *SF* about a particular patient $A$. We assume that the doctor knows the identity $\mathrm{pid}_A$ of the patient, and can thus form the polymorphic pseudonym $\mathrm{ppid}_A = \mathcal{EG}(r, \mathrm{pid}_A, y)$. Figure 2.2 describes the informal descriptions (1.8) and (1.9) in a precise manner.

In Example 2.2.4 we have already seen how *SF* obtains the local pseudonym $\mathrm{pid}_A@SF$. In Figure 2.2 *SF* uses this pseudonym to locate the requested (encrypted) data $\mathcal{EG}(r, D, y)$, and send them back to the Tweaker. These returned data are re-randomised to $E = \mathcal{RR}(\mathcal{EG}(s, D, y), s') = \mathcal{EG}(s' + s, D, y)$, so that they become unlinkable. This Tweaker then re-keys this message $E$ to doctor $B$, who can decrypt the result using his/her own private key $x_B$ since:

$$\begin{array}{rcl} \mathcal{RK}(E, K_B) & = & \mathcal{RK}(\mathcal{EG}(s' + s, D, y), K_B) \\ & \overset{(2.7)}{=} & \mathcal{EG}(\frac{s'+s}{K_B}, D, K_B \cdot y) \\ & = & \mathcal{EG}(\frac{s'+s}{K_B}, D, y_B). \end{array}$$

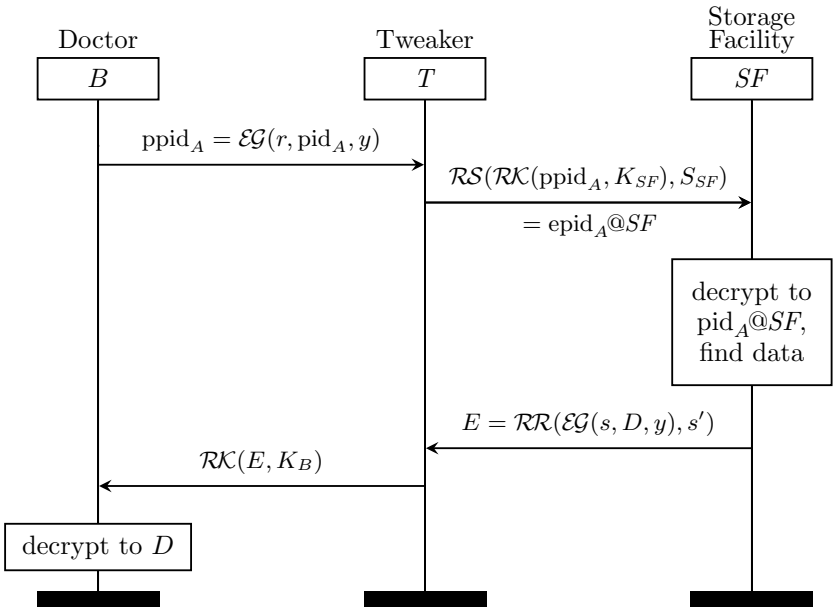



Figure 2.2: Retrieval protocol

Of course, in a realistic version of this protocol additional metadata should be included in the request of the doctor so that the right encrypted data is sent by $SF$. These matters are briefly discussed in the beginning of Section 1.4, see especially Figure 1.2. Additionally, the data request must be logged and checked, to verify that $B$ is entitled to access data about $A$, see Section 2.4 below.

Finally, note that the whole PEP set-up works because private keys and local pseudonyms of participants have a particular form, namely $K \cdot x$ and $S \cdot \mathrm{pid}_A$, using key and pseudonym factors $K$ and $S$. The precise management of these numbers $K$ and $S$ for all participants is a security-critical matter to which we return in Subsection 2.3.1.

## 2.3 Further protocol descriptions

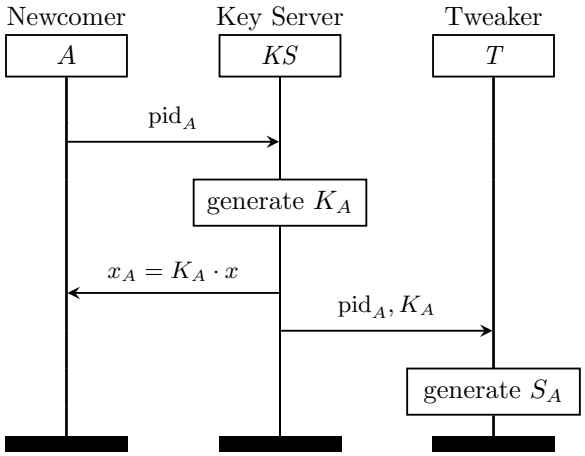
In this section we elaborate some basic protocols that are needed in the deployment of PEP.

### 2.3.1 Key and pseudonimisation factor generation and distribution

As we have seen in the previous section, private keys for the different participants are all derived from a master private key $x$, via scalar multiplication $K \cdot x$. We put this sensitive task in the hands of a separate trusted Key Server $KS$ who is the sole party that possesses the master private key $x$. This private key $x$ is never used for decryption, but is only used to generate diversified keys for the various parties, as will be described below.

(It is possible to split this Key Sever into two parties, where each of them posses a part $x_i$ of the private master key $x = x_1 \cdot x_2$. In this way one can distribute trust, at the expense of simplicity. We shall not follow this idea and use a single Key Server.)

In the descriptions below we implicitly assume that all connections are authenticated and encrypted. We start with a naive protocol that explains what should be achieved.




The numbers $K_A$ and $S_A$ are the key-factor and pseudonym-factor respectively for $A$. They are random numbers in $\mathbb{F}_p$, freshly generated by the Key Server and the Tweaker. These numbers are typically generated in a Hardware Security Module (HSM) as $K_A = KDF(K, \text{`}A\text{'})$, via some key derivation function $KDF$, a master secret $K$, and the identity of $A$. The resulting private key $x_A = K_A \cdot x$ of $A$ must of course be sent to $A$. The Tweaker must know the key factor $K_A$ for $A$, so that it can appropriately re-key messages that $A$ should decrypt. The pseudonym-factor $S_A$ for $A$ only needs to be known to the Tweaker.

An issue with this naive protocol is that $KS$ knows everyone's private key. In a secure implementation this generation of the numbers $K_A, S_A$ is done in an HSM, which reduces the risks. However, there are various ways to improve the situation. For instance the Key Server can be split into two parties, as already mentioned, where each of them contributes a one of $L_1, L_2$ to the key factor $K_A = L_1 \cdot L_2$. We present a different approach in Figure 2.3, where the Tweaker participates in the key generation via a random contribution $R \in \mathbb{F}_p$. In this situation neither the Tweaker nor the Key Server learns the new private key $x_A$ for $A$ — unless they collude.

Of course, the Key Server still owns the super-sensitive master private key $x$ — for the master public key $y = x \cdot g$. Hence $KS$ can decrypt all polymorphic pseudonyms $\text{ppid}_A = \mathcal{EG}(r, \text{pid}_A, y)$. Despite the improvements in Figure 2.3, $KS$ remains a highly trusted party.

### 2.3.2 Patient registers at doctor

We consider the situation where a patient $A$ comes for the first time to a medical doctor $B$ — or some other medical practitioner. In this case the patient $A$ somehow proves that $\text{pid}_A$ is his/her own identifier. Also, the patient $A$ and

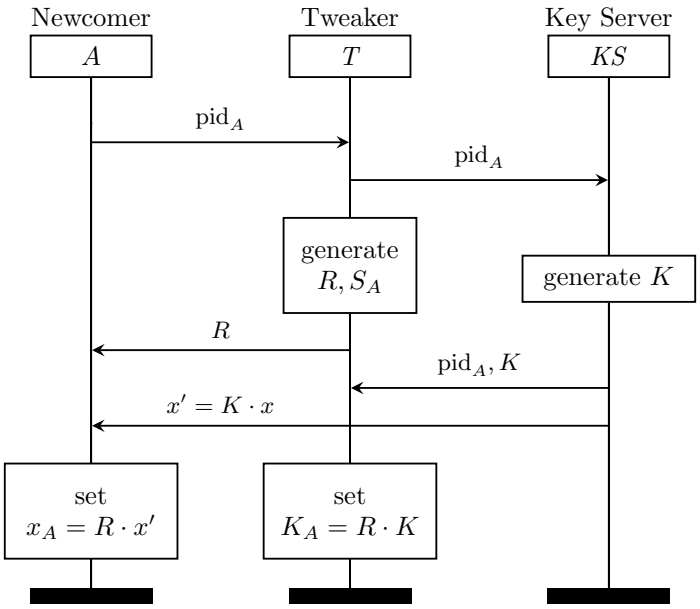
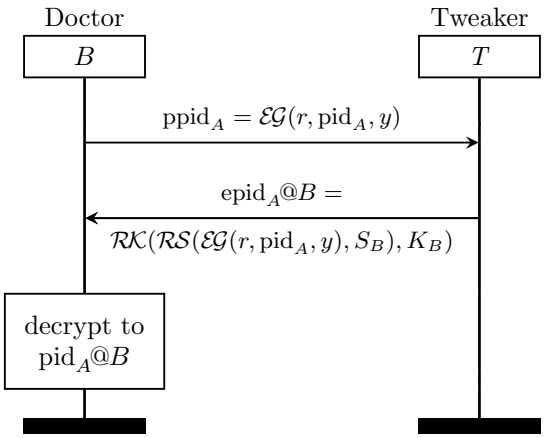



Figure 2.3: Key generation and distribution with participation of the Tweaker

doctor $B$ somehow register an agreement that $B$ gets access to (part of) the existing and new medical data about $A$ that are stored at the Storage Facility $SF$. At this initial stage the doctor not only stores $\mathrm{pid}_A$ in his/her own registration, but also the local pseudonym $\mathrm{pid}_A@B$ of $A$ at the doctor $B$. This pseudonym is needed later, when $B$ needs to link messages with this pseudonym $\mathrm{pid}_A@B$ to the actual identity $\mathrm{pid}_A$, and thus to the patient $A$ — especially in de-pseudonymisation, see the end of Subsection 2.3.3. For this goal the doctor engages in the following one-time interaction with the Tweaker.

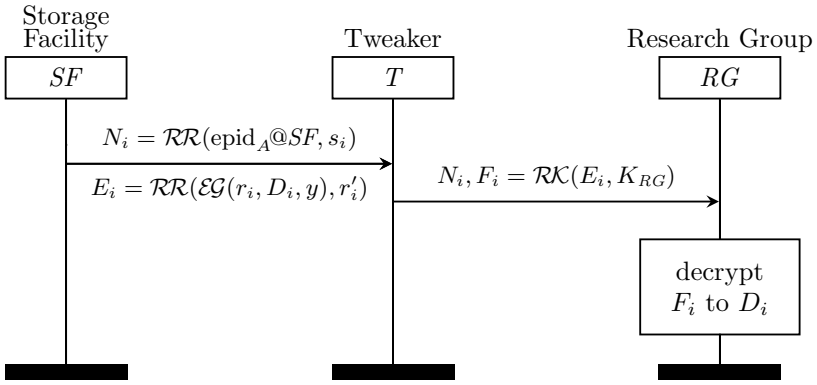



The return message decrypts to $S_B \cdot \mathrm{pid}_A = \mathrm{pid}_A@B$ by (2.9). The Tweaker learns nothing.

As the protocol stands, $B$ can obtain local pseudonyms of everyone of whom (s)he nows the identity pid. The protocol needs to be extended with authentication for $B$. In addition, the pseudonym request must be logged, so that it can be inspected later, see Section 2.4.

### 2.3.3 Research group gets pseudonymised data

We consider the situation where a Research Group $RG$ has submitted a medical research proposal to an oversight committee and that the enclosed request for data has been approved. The Storage Facility then has to provide the relevant data, via the Tweaker. We assume that the Research Group has registered, as in Subsection 2.3.1, and has obtained its own private key $x_{RG} = K_{RG} \cdot x$, where the Tweaker knows the key-factor $K_{RG}$. The relevant encrypted data items that $RG$ should get access to are of the form $\mathcal{EG}(r_i, D_i, y)$, stored at $SF$ under local pseudonym $\mathrm{pid}_{A_i}@SF$, for $i = 1, \ldots, n$. The data delivery protocol then works as follows.

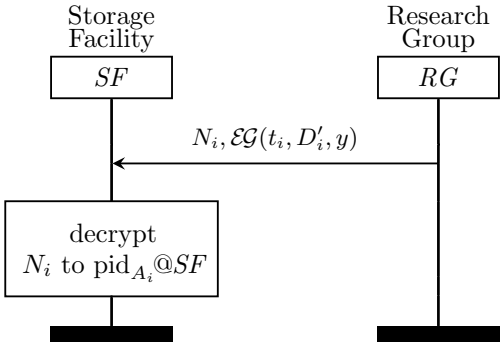



The re-randomisations $N_i$ are still encrypted pseudonyms of $A_i$ at *SF*. In the second step, the re-randomisation and re-keying operations yield a ciphertext:

$$\begin{aligned} F_i \;=\; \mathcal{RK}(E_i, K_{RG}) &= \mathcal{RK}(\mathcal{RR}(\mathcal{EG}(r_i, D_i, y), r'_i), K_{RG}) \\ &= \mathcal{RK}(\mathcal{EG}(r'_i + r_i, D_i, y), K_{RG}) \\ &= \mathcal{EG}(\tfrac{r'_i + r_i}{K_{RG}}, D_i, K_{RG} \cdot y) \\ &= \mathcal{EG}(\tfrac{r'_i + r_i}{K_{RG}}, D_i, y_{RG}). \end{aligned}$$

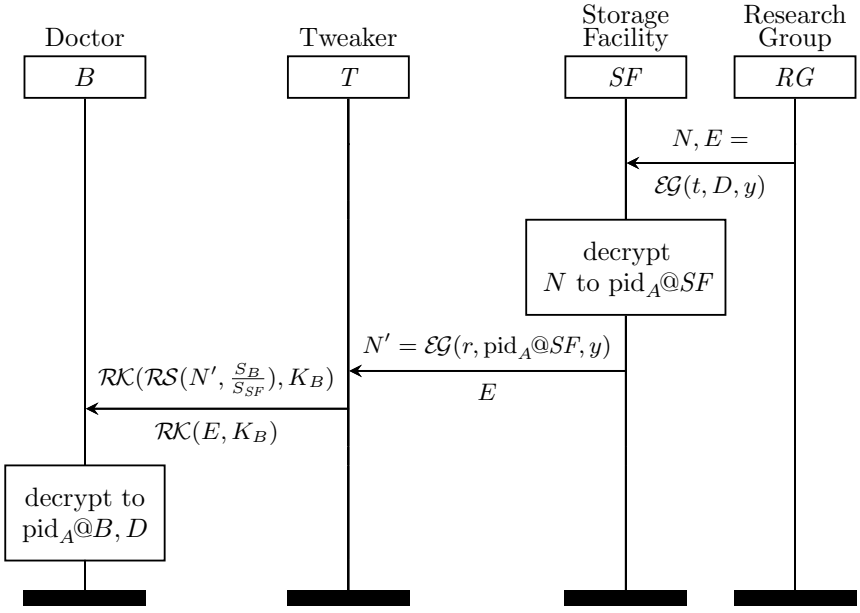
It can thus be decrypted by *RG*. The re-randomised encrypted pseudonyms $N_i$ are used as names/identifiers for the data $D_i$, and are stored together, as pairs $\langle N_i, D_i \rangle$ by *RG*.

Possibly, the Research Group chooses — or has a contractual obligation — to store their findings about the data back into the Storage Facility, and make them accessible by others. This can be done as follows. Let $D'_i$ be the enhanced version of data $D_i$. Using $N_i$ as above, we get:



The Storage Facility can decrypt $N_i = \mathcal{RR}(\mathrm{pid}_{A_i}@SF, s_i)$ with its own private key $x_{SF}$ to the original local pseudonym $\mathrm{pid}_{A_i}@SF$, and store the encrypted enhanced data $\mathcal{EG}(t_i, D_i', y)$ under database key $\mathrm{pid}_{A_i}@SF$, where others can retrieve it. Notice that in this case the Tweaker is not needed as intermediate party.

A more interesting scenario arises when there is a coincidental finding by $RG$, which should be reported back to the doctor and patient. Let's assume this happens in data $D$ for name $N = \mathcal{EG}(s, \mathrm{pid}_A@SF, y_{SF})$. Let's further assume that it is somehow clear from the metadata stored at $SF$ that $B$ is the doctor for patient $A$ and the source of the data $D$. The path back to doctor $B$ then works as follows.




Doctor $B$ can decrypt $\mathcal{RK}(E, K_B)$ to data $D$ by (2.7). But $B$ can also obtain the local pseudonym $\mathrm{pid}_A@B$ for patient $A$ by decrypting:

$$\begin{aligned}
\mathcal{RK}\Big(\mathcal{RS}\big(N', \tfrac{S_B}{S_{SF}}\big), K_B\Big) &= \mathcal{RK}\Big(\mathcal{RS}\big(\mathcal{EG}(r, \mathrm{pid}_A@SF, y), \tfrac{S_B}{S_{SF}}\big), K_B\Big) \\
&= \mathcal{RK}\Big(\mathcal{RS}\big(\mathcal{EG}(r, S_{SF} \cdot \mathrm{pid}_A, y), \tfrac{S_B}{S_{SF}}\big), K_B\Big) \\
&\overset{(2.9)}{=} \mathcal{RK}\Big(\mathcal{EG}(\tfrac{S_B}{S_{SF}} \cdot r, \tfrac{S_B}{S_{SF}} \cdot S_{SF} \cdot \mathrm{pid}_A, y), K_B\Big) \\
&= \mathcal{RK}\Big(\mathcal{EG}(\tfrac{S_B \cdot r}{S_{SF}}, S_B \cdot \mathrm{pid}_A, y), K_B\Big) \\
&\overset{(2.7)}{=} \mathcal{EG}(\tfrac{S_B \cdot r}{K_B \cdot S_{SF}}, \mathrm{pid}_A@B, K_B \cdot y) \\
&= \mathcal{EG}(\tfrac{S_B \cdot r}{K_B \cdot S_{SF}}, \mathrm{pid}_A@B, y_B).
\end{aligned}$$

By this 'EP to EP' conversion, Doctor $B$ gets his/her own local pseudonym $\text{pid}_A@B$ and can find the corresponding patient identifier $\text{pid}_A$ in his/her own administration, via the stored link to the local pseudonym $\text{pid}_A@B$, see Subsection 2.3.2; $B$ can then contact patient $A$ about the reported findings in $D$. Notice that the other parties in the above protocol do not know the identity of patient $A$.

The 'EP to EP' conversion can be avoided if the data record of patient $A$ also stores for each contributing doctor $B$ also his/her local pseudonym $\text{pid}_A@B$ of $A$ at $B$. The Storage Facility can then simply include this pseudonym when it reports the coincidental finding back to doctor $B$.

### 2.3.4 Data Sanitation

The data stemming from current wearable devices are sent to the database of the operator. The data are typically encrypted during transfer. Once transferred, they are made available again to the owner of the device, via some webinterface. The stored data can also be accessed by the operator, for various additional services and other commercial purposes — such as sale to third parties. There is often one good reason why the operator needs to get access to the 'raw' data from the device, namely in order to *sanitise* the data. Such sanitisation may involve consistency checks or improvements which are computationally too intensive to perform on the wearable device itself.

In the PEP setup all (sensitive) data from a wearable device are immediately encrypted on the device, and sent to a storage provider (via the Tweaker), see Example 2.2.2, or the informal description in (1.6). Can the operator then still sanitise the data, if needed?

Sanitation is stil possible in a PEP framework via protocols like in Subsection 2.3.3, where a research group gets access to selected pseudonymised data $D$, and returns an 'improved' version $D'$. This $D'$ can also be a sanitised version of $D$. By doing such sanitations in a batchwise manner the operator can still get statistical information about certain data (groups), but cannot get information about identifiable users. Such a set-up is only possible in close cooperation with the device manufacturers/operators. They may be interested to adapt their devices to a PEP framework for instance under pressure from health care providers, from consumers or from data protection regulators, or simply because they themselves wish to operate in a privacy-friendly manner.

## 2.4 Authentication, authorisation, and logging

So far we have concentrated on protocols for the basic PEP functionality. In a wider eco-system these protocols will have to be complemented with mechanisms for authentication, authorisation and logging. These topics will be addressed in the current section. We start with a description of the issues involved, and continue with a more operational description.

**Authentication** We use the word 'authentication' for the procedure of proving certain properties about oneself. Authentication may apply to both humans and to computers. In a narrow sense authentication is about proving your identity, for instance by showing your passport, or by provinding the right password or PIN — proving that you are identified by the corresponding login name, or bank account number.

In the PEP framework there is an obvious necessity to authenticate, for instance when a patient wants to see his/her own files, or when a medical doctor or researcher needs to access certain files with personal information. The above description uses an abstract personal identifier $\text{pid}_A$ of person $A$. Concretely, the authentication goal is to prove that $\text{pid}_A$ really belongs to $A$.

Authentication of devices is also an issue in the PEP framework, especially for wearables for self-measurement. On the one hand, some level of authentication is needed, to exclude arbitrary devices sending data into the system, which could lead to overload and denial-of-service (DoS). On the other hand, the authentication requirements for devices should not undermine pseudonymisation.

In identity management one distinguishes different quality assurance levels for authentication, as developed for instance in the context of STORK[3] or ISO/IEC 29115[4]. These two frameworks each have four levels of 'entity authentication assurance', roughly indicated as 'low' (level 1), 'medium' (level 2), 'high' (level 3), 'very high' (level 4). In health care one typically requires the last two levels of assurance.

It may be useful to build authentication not only on a single identifier, like pid, but to use more general attributes, like 'medical doctor', 'assessor', 'nurse', possibly extended with medical areas of expertise. Such attribute-based authentication is seen as a promising direction in identity management, see for instance the work on U-prove [3] or Idemix [5, 2, 1], or relevant standardisation efforts[5].

**Authorisation** The process of authorisation tells whether an already authenticated user is permitted to have access to a certain resource. Roughly, one can distinguish 'read' access from 'write' access. This 'write' access can be divided further into 'modification', or 'addition-only'; the latter form preserves the history of a file, and is sometimes preferred in a medical setting. Typically, nurses and patients only have read access to a personal medical dosssier, and only doctors have write access.

Authorisation is not a straightforward matter.

- Obviously, patients should be permitted to read their own dossier. But in some cases person $A$ should also be permitted to read the dossier of person $B$, for instance when $B$ is a minor and $A$ is a parent or legal custodian.

[3] See `www.eid-stork.eu/`

[4] See `www.iso.org/iso/catalogue_detail.htm?csnumber=45138`

[5] For instance by NIST, see `https://nccoe.nist.gov/projects/building_blocks/attribute_based_access_control`

Such a guardianship may also occur in other situations, like when someone suffers from dementia or Alzheimer.

- Access to a dossier by medical staff should be permitted only if there is a 'treatment relation'. But it is surprisingly hard to formalise such a relationship, for instance because of (temporary) absence, second opinions, external advice or assistance, *etc.* In practice it has become clear that formal authorisation rules are too rigid. Instead, one sometimes sees medical systems giving a warning or asking for explicit motivations in situations that look non-standard. At the same time each dossier access is logged.

**Logging** The activity of recording each event or transaction of a certain kind in a special 'log' file is called logging. It is useful for abuse detection and for reconstructing what went wrong after an incident. Logging in privacy-friendly systems is a delicate matter, since confidentiality should be maintained. One possible solution is to cryptographically protect the log files and to grant access only to a trusted party — only under specific circumstances, or only on a statistical basis.

In the PEP set-up we aim to give each participant $A$ access to the transactions in the log file in which $A$ occurs. In particular, this means that patients get to see who has accessed their dossiers at which stage, possibly also with which purpose. This is done primarily for transparancy reasons, but may also be useful for anomaly detection.

In the remainder of this section we give two protocols in which authentication and logging are made explicit. This will happen via two new trusted parties, called the Access Manager $AM$ and the Logger $L$.

We first reconsider the retrieval of patient data by a doctor. The extended version of the earlier protocol in Figure 2.2 is described in Figure 2.4. We shall step through this protocol. First, Doctor $B$ authenticates him/herself to the Access Manager. The authentication message uses $\mathrm{pid}_B$ as personal identifier, which, in this case, is typically a medical registration numbers. This yields some ticket, which is not written explicitly[6], that is used in the next step, where the doctor sends his/her own identity $B$, the query $Q$, and the polymorphic pseudonym $\mathrm{ppid}_A = \mathcal{EG}(r, \mathrm{pid}_A, y)$ to the Tweaker. The Tweaker does two things.

- It applies the PP2EP conversion to the polymorphic pseudonym $\mathrm{ppid}_A$, giving the encrypted pseudonym $\mathrm{epid}_A@L$ for the Logger. It then passes on the triple $B, Q, \mathrm{epid}_A@L$ to the Logger. This Logger can decrypt $\mathrm{epid}_A@L$ to the local pseudonym $\mathrm{pid}_A@L$, which is stored together with $B$ and $Q$ as retrieval request in the log file. Optionally, the identity $B$ of the doctor is not logged in the clear, but also in encrypted form, as polymorphic pseudonym $\mathrm{ppid}_B = \mathcal{EG}(s, \mathrm{pid}_B, y)$.

[6] This ticket is also used to link the various messages in the protocol.

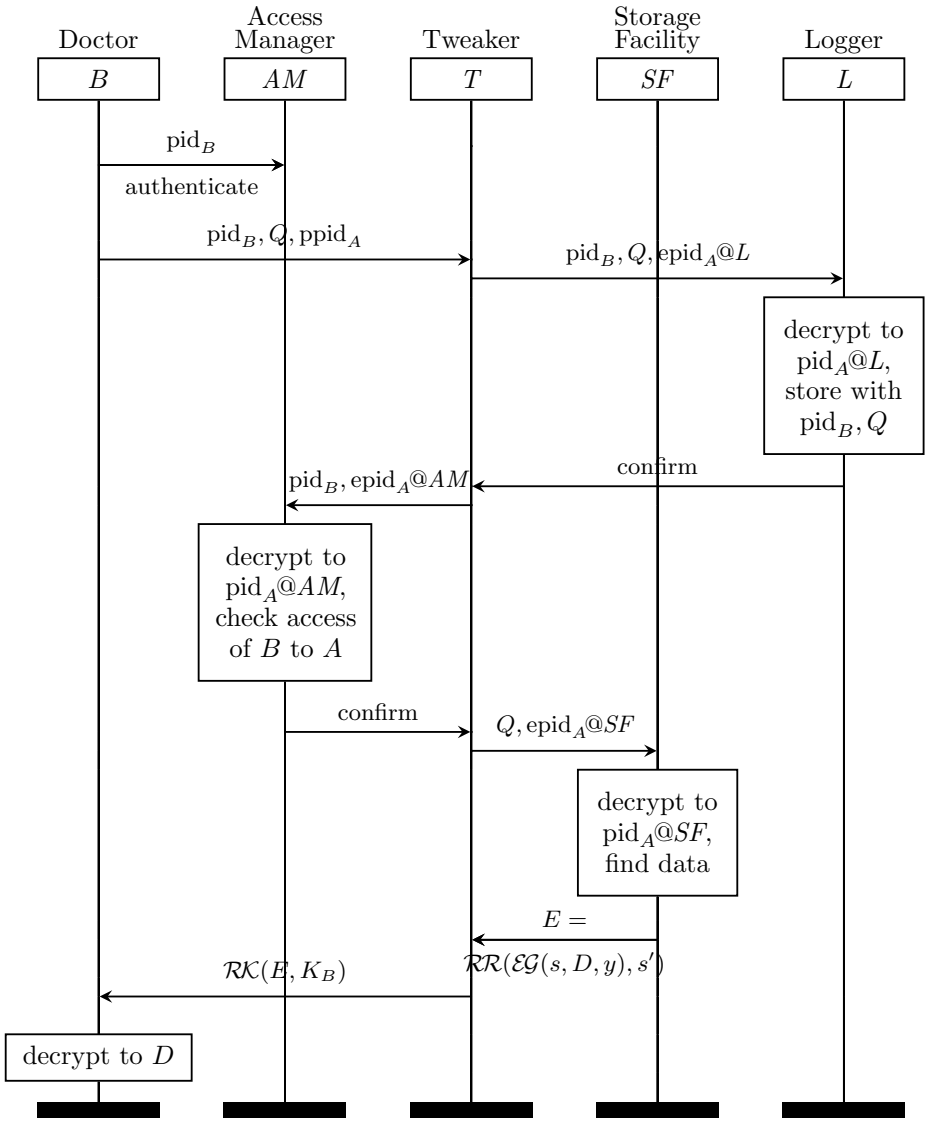



Figure 2.4: Retrieval protocol from Figure 2.2 extended with access control and logging

- The Tweaker similarly turns the polymorphic pseudonym $\text{ppid}_A$ into an encrypted pseudonym $\text{epid}_A@AM$ of $A$ for the Access Manager. The latter can decrypt $\text{epid}_A@AM$ to the local pseudonym $\text{pid}_A@AM$ of $A$, and check if there are rules that permit/exclude access of $B$ to $A$. Notice that these rules will have to be formulated in terms of local pseudonyms of patients, so that the Access Manager does not learn the identities of the patients involved. Optionally, the query $Q$ can also be sent to the Access Manager, to be part of the decision process. (In this set-up we assume that the Access Manager does not keep any logs.)

After an (authenticated) confirmation message from the Access Manager, the Tweaker proceeds to compute the encrypted pseudonym $\text{epid}_A@SF$ of $A$ at the Storage Facility and to pass it on together with the original query $Q$. Notice that the Storage Facility learns nothing about the source $B$ of the query. It decrypts the encrypted pseudonym, finds the associated database record, and selects the relevant encrypted data using the query $Q$. These encrypted data are of the form $\mathcal{EG}(s, D, y)$, and are passed on to the Tweaker, in re-randomised form, who re-keys them to Doctor $B$, like in the first version of the protocol in Figure 2.2. In total, the Tweaker performs three PP2EP conversions. In this protocol the Storage Facility is trusted to return the right encrypted data, corresponding to query $Q$. We shall reconsider this assumption in Subsection 2.5.1.

Our second protocol is inspection of log files by a participant $A$, see Figure 2.5. After authentication, the Access Manager passes the identity $A$ on to the Tweaker. It produces an encrypted pseudonym for the Logger, who can decrypt the latter to its own local pseudonym $\text{pid}_A@L$ for $A$. It logs the request itself, and finds all log entries $E_i$ in which the local pseudonym occurs. These entries are returned to the Tweaker, in polymorphically encrypted form. The Tweaker remembers that they should be re-keyed to $A$, and passes them on. In this way the Logger does not learn who requests the log entries, and the Tweaker cannot read the selected entries.

The protocol in Figure 2.5 only describes inspection by individuals. Inspection may also be performed by health care or data protection authorities. They may ask different questions, like: about which individuals did health care professional $X$ retrieve which data? The Logger can only answer with local pseudonyms $\text{pid}_A@L$ of individuals $A$. These pseudonyms may be translated by the Tweaker to pseudonyms of the inspection authorities, via an 'EP to EP' conversion like in Subsection 2.3.3. If the inspection authorities have access to the real identities pid, they can find out who actually occurred in the log files — if needed.

## 2.5 Cryptographic enhancements

The previous sections have described the main lines of the PEP framework. In a complicated system like this, many variations are possible. The decision which of them should be implemented depends on many factors, like complexity,

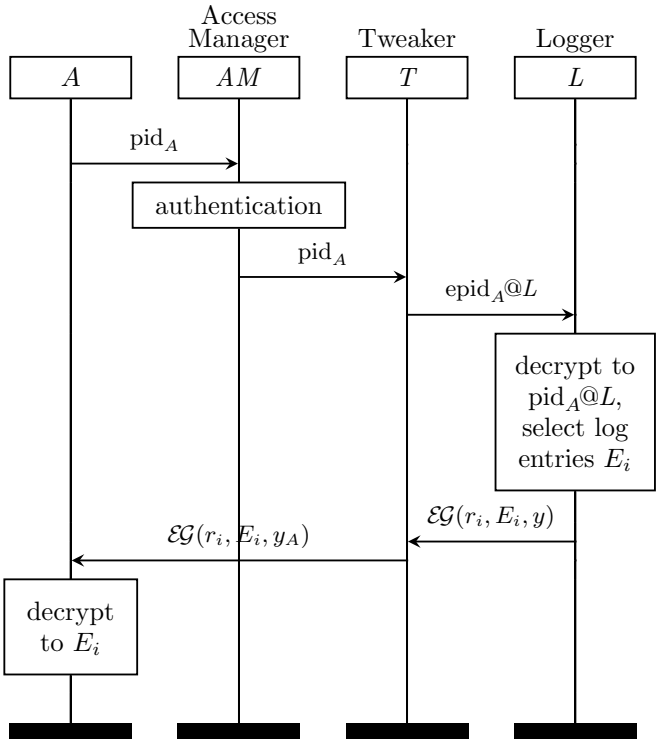



Figure 2.5: Protocol for obtaining relevant log file entries

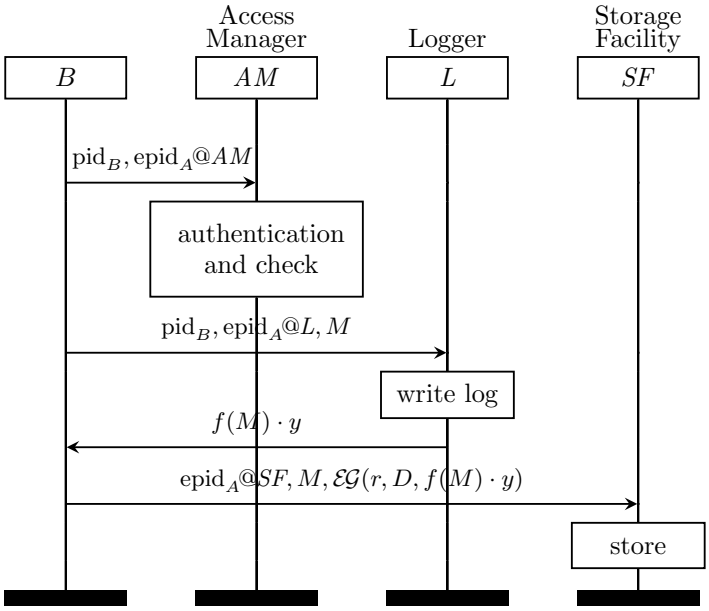



Figure 2.6: Protocol for storing encrypted data, linked to metadata

computational overhead, perceived risks, and added guarantees. This section describes two of such possible variations.

### 2.5.1 Linking metadata and data

Our first variation addresses the following point, which was briefly mentioned in the description of the retrieval protocol of Figure 2.4. Given a query for data, we have to trust the Storage Facility to return the right (encrypted) data, associated with the query. For instance, a malfunctioning or malicious Storage Facility could return DNA data when only (body) weight is requested.

A possible solution is to link the metadata, occurring in a query, to the data itself. In storage protocols, like in Example 2.2.4, we have stored data $D$ via polymorphic encryption as $\mathcal{EG}(r, D, y)$, using the master public key $y$. Let $M$ be the metadata for $D$, describing for instance the source, date, label, format *etc.* of $D$, see Figure 1.2. Our variation is to encrypt $D$ as $\mathcal{EG}(r, D, f(M) \cdot y)$ with a public key $f(M) \cdot y$ which depends on the metadata, via some function $f$. One can think of $f$ as some keyed hash function, but what it precisely does is not relevant.

The important point is: if the Storage Facility returns the wrong encrypted

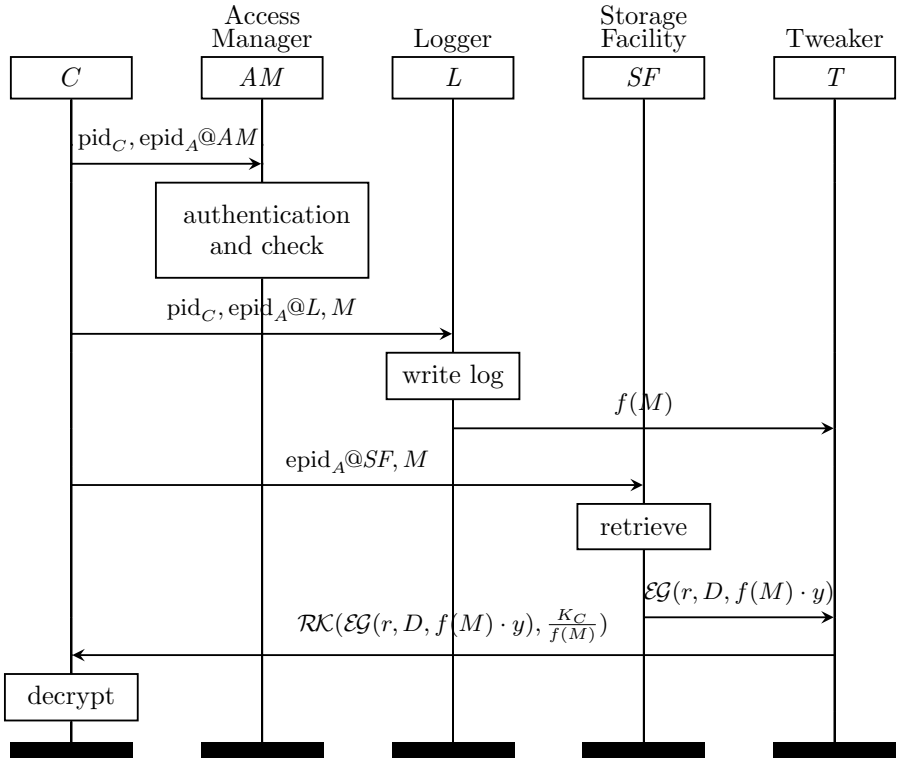



Figure 2.7: Protocol for retrieving encrypted data, linked to metadata

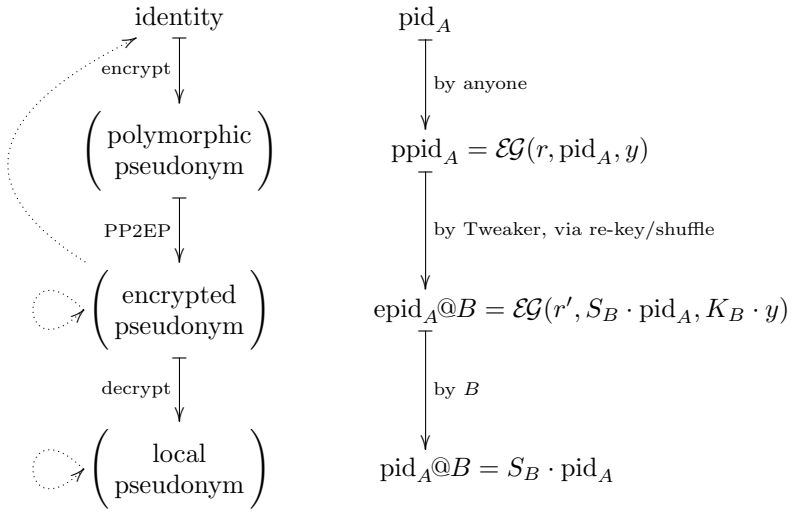
data $\mathcal{EG}(r, D', f(M') \cdot y)$, not belonging to metadata $M$, then this ciphertext cannot be decrypted with the private key belonging to the public key $f(M) \cdot y$, associated with the right metadata.

What matters is who computes this function $f$. We choose to let the Logger do this, because the Logger also registers the metadata $M$ in a storage or retrieval protocol. Our adapted protocols, are described in Figures 2.6 and 2.7. In the first, storage protocol we have left out the Tweaker, for reasons of simplicity. It should provide the encrypted pseudonyms epid$_A$@$X$, for the various parties $X$. In the retrieval protocol the Tweaker plays the important role of re-keying, with factor $\frac{K_C}{f(M)}$. It turns the encryption of $D$ with public key $f(M) \cdot y$ into an encryption with public key $K_C \cdot y = y_C$, so that $C$ can decrypt and obtain $D$.

### 2.5.2 Separating polymorphic and encrypted pseudonyms

We recall from Figure 1.1 that for a participant with identity $\mathrm{pid}_A \in \mathbb{G}$, a *polymorphic pseudonym* is an ElGamal encryption $\mathrm{ppid}_A = \mathcal{EG}(r, \mathrm{pid}_A, y)$ with the master public key $y$. An *encrypted pseudonym* for participant $B$ is of the form $\mathrm{epid}_A@B = \mathcal{EG}(r, \mathrm{pid}_A@B, y_B)$, where $\mathrm{pid}_A@B = S_B \cdot \mathrm{pid}_A \in \mathbb{G}$ is the local pseudonym of $A$ for $B$, and $y_B = K_B \cdot y$ is the public key of $B$. This encrypted pseudonym $\mathrm{epid}_A@B$ can be obtained from the polymorphic pseudonym $\mathrm{ppid}_A$ via re-keying and re-shuffling, called PP2EP conversion, see Definition 2.2.5.

We given an overview of the different conversions.




(2.13)

The diagram on the left contains three additional, dotted arrows, representing the following options.

1. An encrypted pseudonym $\mathrm{epid}_A@B$ for $B$ can be turned into an encrypted pseudonym $\mathrm{epid}_A@C$ for another participant $C$. The Tweaker can do this via re-shuffling with factor $S_C/S_B$, like in the 'EP to EP' conversion in Subsection 2.3.3.

2. Also, an encrypted pseudonym $\mathrm{epid}_A@B$ can even be turned into the identity $\mathrm{pid}_A$ if the Tweaker re-shuffles it with factor $1/S_B$. It yields $\mathcal{EG}(r, \mathrm{pid}_A, y_B)$, which can be decrypted by $B$.

3. Similarly, the tweaker can transform a pseudonym $\mathrm{pid}_A@B$ into $\mathrm{pid}_A@C$ by using the appropriate pseudonymization factors.

One may argue that these two points give too much flexibility, and that only the downward direction should be allowed in Diagram 2.13. The 'EP to EP' conversion in Subsection 2.3.3 should then be avoided, for instance by keeping a polymorphic pseudonym $\mathrm{ppid}_A$ of patient $A$ in its database record — in addition to the local pseudonym $\mathrm{pid}_A@SF$ that is used as key of the record.

We present a way to enforce this downward-direction-only policy by using a bilinear pairing. Such a pairing is a function of the form:

$$\mathbb{G} \times \mathbb{H} \xrightarrow{\ e\ } \mathbb{K}$$

where $\mathbb{G}, \mathbb{H}$ and $\mathbb{K}$ are three (additive) groups. The crucial property of the pairing map $e$ is that the functions $e(-,b)\colon \mathbb{G} \to \mathbb{K}$ and $e(a,-)\colon \mathbb{H} \to \mathbb{K}$ are group homomorphisms. That is,

$$\begin{aligned} e(0,b) &= 0 & e(a,0) &= 0 \\ e(a+a',b) &= e(a,b) + e(a',b) & e(a,b+b') &= e(a,b) + e(a,b'), \end{aligned}$$

for all $a, a' \in \mathbb{G}$ and $b, b' \in \mathbb{H}$. As a result one has:

$$e(n \cdot a, m \cdot b) = n \cdot m \cdot e(a,b),$$

for all group elements $a \in \mathbb{G}, b \in \mathbb{H}$ and numbers $n, m \in \mathbb{Z}$. In the present context we further assume that the groups $\mathbb{G}$ and $\mathbb{H}$ are generated by elements $g, h$, so $\mathbb{G} = \langle g \rangle$ and $\mathbb{H} = \langle h \rangle$, where $g, h$ are of the same prime order $p$. In addition, we assume that $\mathbb{K}$ has order $p$, and that the element $e(g,h) \in \mathbb{K}$ is a generator, hence non-zero and of order $p$. The Barreto-Naehrig curve BN254 [18] provides an example. We shall make special use of the group isomorphism[7]:

$$\mathbb{G} \xrightarrow[\cong]{i \stackrel{\text{def}}{=} e(-,h)} \mathbb{K}$$

Below we write $i^3\colon \mathbb{G}^3 \to \mathbb{K}^3$ for the function that sends a triple $\langle a,b,c \rangle$ to $\langle i(a), i(b), i(c) \rangle$ by applying $i$ coordinatewise. This notation $i^3$ is thus not used for iteration.

We still assume a master public key $x \in \mathbb{F}_p$ with associated public key $y = x \cdot g \in \mathbb{G}$. Each participant $B$ still has a private key $x_B = K_B \cdot x$. However the corresponding public key $y'_B$ no longer resides in the group $\mathbb{G}$ but in $\mathbb{K}$. It is the $i(.)$ image of the original public key $y_B = x_B \cdot g$. That is:

$$y'_B \stackrel{\text{def}}{=} i(y_B) = i(x_B \cdot g) = x_B \cdot i(g) = x_B \cdot e(g,h) \in \mathbb{K}. \tag{2.14}$$

The participants no longer need the original public keys $y_B \in \mathbb{G}$; these original public keys are only used implicitly, to form their $i(.)$ image. Identities $\mathrm{pid}_A$ are still group elements in $\mathbb{G}$, like before, with polymorphic pseudonyms $\mathrm{ppid}_A = \mathcal{EG}(r, \mathrm{pid}_A, y) \in \mathbb{G}^3$.

Our set-up now further changes in the follow way. First of all, the Tweaker is no longer provided with the original key factor $K_B$ of a participant $B$ but only with a suitably encrypted variant of it. To this end, the Tweaker is given $K_B \cdot h \in \mathbb{H}$ instead. Moreover, the original pseudonym factor $S_B$ of a participant $B$ is also provided in an encrypted fashion, in a such a form that no party, even

[7] We only use that $i$ is an injection.

the Tweaker, has possession of it — so that, as a result, the 'dangerous' dotted arrows in Diagram (2.13) disappear. To this end, the Tweaker simply selects a random element $h_B \in \mathbb{H}$. This implicitly defines a pseudonym factor $S_B$ as the discrete logarithm of $h_B$ with respect to $h$. Below, we will still use the pseudonym factor $S_B$ but in only in an implicit way where it suffices to have $h_B$.

Our set-up now changes in the follow way.

1. We keep the same notation $\mathrm{pid}_A@B = S_B \cdot \mathrm{pid}_A \in \mathbb{G}$ for the original local pseudonym. The new version $\mathrm{pid}_A@'B$ is written with a prime ($'$) and is now an element of the group $\mathbb{K}$, defined as:

$$\begin{aligned} \mathrm{pid}_A@'B &\stackrel{\text{def}}{=} e(\mathrm{pid}_A, h_B) \\ &= e(\mathrm{pid}_A, S_B \cdot h) \qquad \text{assuming } h_B = S_B \cdot h \\ &= S_B \cdot e(\mathrm{pid}_A, h) \\ &= e(S_B \cdot \mathrm{pid}_A, h) \\ &= e(\mathrm{pid}_A@B, h) \\ &= i\big(\mathrm{pid}_A@B\big). \end{aligned}$$

   Hence the isomorphism $i\colon \mathbb{G} \to \mathbb{K}$ maps the old local pseudonym to the new one — if we know that $S_B$ is the discrete log of $h_B$.

   The Tweaker is the only party that can compute such local pseudonyms. In the original set-up, the Tweaker had to know the pseudonymisation factor $S_B \in \mathbb{F}_p$. In the current set-up nobody needs to know the number $S_B$. That is, the Tweaker only needs to posses a randomly selected element $h_B \in \mathbb{H}$. Typically this generated by the Tweaker using a secret embedding into the group $\mathbb{H}$, based on the identity of participant B and a secret diversification key.

2. Recall that the original encrypted pseudonym is defined as:

$$\mathrm{epid}_A@B = \mathcal{EG}\big(r, \mathrm{pid}_A@B, y_B\big) \in \mathbb{G}^3.$$

   The new version $\mathrm{epid}_A@'B \in \mathbb{K}^3$ is defined as an ElGamal encryption in $\mathbb{K}$, via:

$$\mathrm{epid}_A@'B \stackrel{\text{def}}{=} \mathcal{EG}\big(r,\, \mathrm{pid}_A@'B,\, y'_B\big) \quad \text{where} \quad y'_B = i(y_B) \quad \text{see (2.14)}.$$

   We claim that the map $i\colon \mathbb{G} \to \mathbb{K}$ again preserves this structure, via the equation: $i^3\big(\mathrm{epid}_A@B\big) = \mathrm{epid}_A@'B$. Indeed, using $i(g) = e(g, h) \in \mathbb{K}$ as

generator, we have:

$$\begin{aligned}
\mathrm{epid}_A@'B &= \mathcal{EG}\big(r,\, \mathrm{pid}_A@'B,\, y'_B\big) \\
&= \langle\, r \cdot i(g),\, r \cdot y'_B + \mathrm{pid}_A@'B,\, y'_B \,\rangle \\
&= \langle\, i(r \cdot g),\, r \cdot i(y_B) + i(\mathrm{pid}_A@B),\, i(y_B) \,\rangle \\
&= \langle\, i(r \cdot g),\, i(r \cdot y_B + \mathrm{pid}_A@B),\, i(y_B) \,\rangle \\
&= i^3\langle\, r \cdot g,\, r \cdot y_B + \mathrm{pid}_A@B,\, y_B \,\rangle \\
&= i^3\big(\mathcal{EG}(r, \mathrm{pid}_A@B, y_B)\big) \\
&= i^3\big(\mathrm{epid}_A@B\big).
\end{aligned}$$

The definitions of re-keying and re-shufflying have to be adapted, and also their combination as in (2.10). The latter combination is needed since re-keying and re-shufflying act 'from $\mathbb{G}$ to $\mathbb{K}$' and there is no way back. So if we wish to combine them, we have to do it immediately. We shall use primed notation ($'$) for the new versions.

**Definition 2.5.1.** *In the above setting, re-keying and re-shuffling and their combination are defined as functions of type:*

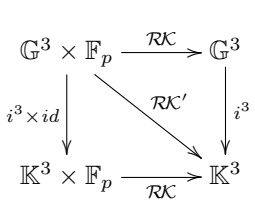
$$\mathbb{G}^3 \times \mathbb{F}_p \xrightarrow{\mathcal{RK}'} \mathbb{K}^3 \qquad \mathbb{G}^3 \times \mathbb{H} \xrightarrow{\mathcal{RS}'} \mathbb{K}^3 \qquad \mathbb{G}^3 \times \mathbb{F}_p \times \mathbb{H} \xrightarrow{\mathcal{RKS}'} \mathbb{K}^3$$

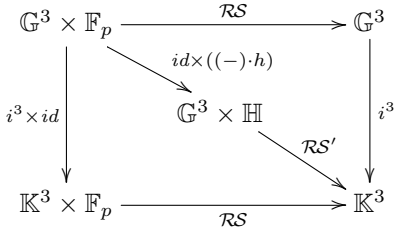
*Explicitly:*

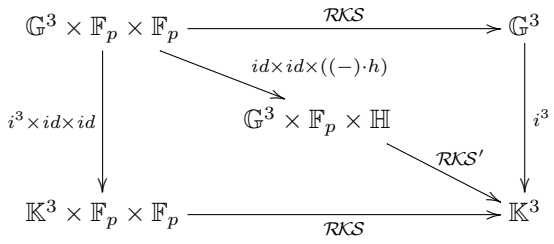
$$\begin{aligned}
\mathcal{RK}'(\langle b, c, y\rangle, k) &\stackrel{def}{=} i^3\big(\mathcal{RK}(\langle b, c, y\rangle, k)\big) \;=\; \langle\, i(\tfrac{1}{k} \cdot b),\, i(c),\, i(k \cdot y) \,\rangle \\
\mathcal{RS}'(\langle b, c, y\rangle, z) &\stackrel{def}{=} \langle\, e(b, z),\, e(c, z),\, i(y) \,\rangle \\
\mathcal{RKS}'(\langle b, c, y\rangle, k, z) &\stackrel{def}{=} \langle\, e(\tfrac{1}{k} \cdot b, z),\, e(c, z),\, i(k \cdot y) \,\rangle.
\end{aligned}$$

We summarise the situation in terms of commuting diagrams.

**Lemma 2.5.2.** *The new (primed) re-keying and re-shuffling operations from Definition 2.5.1 make the following three diagrams commute.*



*We recall that the original function $\mathcal{RKS}$ is defined in* (2.10).

**Proof** We start with the upper left rectangle. Its upper right triangle commutes by definition of $\mathcal{RK}'$. Its lower left triangle commutes because:

$$\begin{aligned}
\mathcal{RK}\big(i^3\langle b,c,y\rangle,k\big) &= \mathcal{RK}\big(\langle i(b),i(c),i(y)\rangle,k\big)\\
&= \langle\, \tfrac{1}{k}\cdot i(b),\, i(c),\, k\cdot i(y)\,\rangle\\
&= \langle\, i(\tfrac{1}{k}\cdot b),\, i(c),\, i(k\cdot y)\,\rangle\\
&= i^3\langle\, \tfrac{1}{k}\cdot b,\, c,\, k\cdot y\,\rangle\\
&= i^3\big(\mathcal{RK}(\langle b,c,y\rangle,k)\big)\\
&= \mathcal{RK}'(\langle b,c,y\rangle,k).
\end{aligned}$$

We turn to the upper right rectangle. It states that if the element $z\in\mathbb{H}$ is of the form $z=n\cdot h$, then the map $i\colon \mathbb{G}\to\mathbb{K}$ also commutes with re-shuffling, as expressed by the two sub-triangles. They commute since:

$$\begin{aligned}
\mathcal{RS}'(\langle b,c,y\rangle,n\cdot h) &= \langle\, e(b,n\cdot h),\, e(c,n\cdot h),\, i(y)\,\rangle\\
&= \langle\, n\cdot e(b,h),\, n\cdot e(c,h),\, i(y)\,\rangle\\
&= \langle\, n\cdot i(b),\, n\cdot i(c),\, i(y)\,\rangle\\
&= \mathcal{RS}(\langle i(b),i(c),i(y)\rangle,n)\\
&= \mathcal{RS}(i^3\langle b,c,y\rangle,n)\\
\mathcal{RS}'(\langle b,c,y\rangle,n\cdot h) &= \langle\, e(b,n\cdot h),\, e(c,n\cdot h),\, i(y)\,\rangle\\
&= \langle\, n\cdot e(b,h),\, n\cdot e(c,h),\, i(y)\,\rangle\\
&= \langle\, e(n\cdot b,h),\, e(n\cdot c,h),\, i(y)\,\rangle\\
&= \langle\, i(n\cdot b),\, i(n\cdot c),\, i(y)\,\rangle\\
&= i^3\langle\, n\cdot b,\, n\cdot c,\, y\,\rangle\\
&= i^3(\mathcal{RS}(\langle b,c,y\rangle,n)).
\end{aligned}$$

Commutation of the third diagram is left to the interested reader. □

In this new set up, participant $A$ has one private key $x_A\in\mathbb{F}_p$ with *two* associated public keys, in $\mathbb{G}$ and in $\mathbb{K}$, namely:

$$y_A = x_A\cdot g\in\mathbb{G}\quad\text{ and }\quad i(y_A)=x_A\cdot i(g)=x_A\cdot e(g,h)\in\mathbb{K}.$$

Encryptions of data are done in $\mathbb{G}$, and encryptions of pseudonyms in $\mathbb{K}$. A polymorphic pseudonym $\mathrm{ppid}_A=\mathcal{EG}(r,\mathrm{pid}_A,y)$ is an element of $\mathbb{G}^3$. By re-keying-and-shuffling $\mathcal{RKS}'$ it yield an encrypted pseudonym $\mathrm{epid}_A@'B=\mathcal{EG}(r,\mathrm{pid}_A@'B,i(y_B))$ in $\mathbb{K}^3$. It is produced by the Tweaker via its key and pseudonym factors $K_B\in\mathbb{F}_p$ and $h_B\in\mathbb{H}$. In this new set up, the dashed arrows in Diagram 2.13 no longer exist. But the basic PEP functionality can still be provided.

What has been achieved is a limitation of the power of the Tweaker, at the expense of greater cryptographic complexity, via bilinear pairing, and of stronger security assumptions. One can ask if this is worth it, certainly if the Tweaker in its original form is implemented in an HSM. Possible abuse is then controlled by using restricted, secure hardware.

## 2.6 Ongoing work

This white paper describes the basics of Polymorphic Encryption and Pseudonymisation (PEP). This is an ungoing project involving, research, design, development, and deployment. This section gives an impression of some of the ongoing activities. It will be expanded in later versions of this document.

### 2.6.1 Informal security analysis

In this subsection we will present an informal security analysis of the basic protocol (Section 2.2 and 2.3), where we look at how things can go wrong if the different parties are compromised or if they collude.

The Key Server contains the most sensitive components of the system: the master private key $x$. Using this key, all other private keys are generated. Therefore, if the Key Server is compromised possibly all data and pseudonyms could be recovered, via decryption of polymorphic pseudonyms $\mathcal{EG}(r, \text{pid}, y)$ and polymorphically encrypted data $\mathcal{EG}(r, D, y)$ produced by wearables, for instance.

If in addition the Tweaker gets compromised, and key and pseudonym factors become known, all encryptions with participant keys $y_A$ can be undone.

The Tweaker plays an important role in the system as it cryptographically regulates access to the data — via tweaking of keys. In order to do this it knows all key and pseudonym factors. When the Tweaker colludes with just one participant $A$, the master private key $x$ can already be learned using the participant's private key $x_A$ and key factor $K_A$ as follows: $K_A^{-1} \cdot x_A = K_A^{-1} \cdot K_A \cdot x = x$. Now, they can construct all private keys and are able to decrypt any data or pseudonyms. As the Tweaker knows all pseudonym factors, it is also possible to retrieve all the original identities pid. The Tweaker can also tweak all data and pseudonyms to be used by participant $A$. Original identities can be learned by not using a pseudonym factor for the recipient. For example, assume the Tweaker is again colluding with participant $A$, and together they have the pseudonym $\text{epid}_C@B$. First the Tweaker takes out the pseudonym factor of $B$:

$$\begin{aligned}
\mathcal{RS}(\text{epid}_C@B, S_B^{-1}) &= \mathcal{RK}(\mathcal{EG}(r, \text{pid}_C@B, y_B), S_B^{-1}) \\
&= \mathcal{RS}(\mathcal{EG}(r, S_B \cdot \text{pid}_C, y_B), S_B^{-1}) \\
&= \mathcal{EG}(r \cdot S_B^{-1}, S_B \cdot \text{pid}_C \cdot S_B^{-1}, y_B) \\
&= \mathcal{EG}(r \cdot S_B^{-1}, \text{pid}_C, y_B)
\end{aligned}$$

Now the Tweaker can generate $B$'s private key to decrypt, or, alternatively, re-key this result for participant A.

| **Colluding parties** | **Result** |
|---|---|
| Key Server and Tweaker | All private keys can be retrieved, so all data and identities can be accessed |
| Access Manager and a researcher | This researcher can get access to all pseudonymised data |
| Access Manager and a doctor | This doctor can get access to all pseudonymised data and can re-identify his/her own patients |
| Tweaker and a participant | The master private key can be retrieved, and all other private keys too, so all data and identities can be accessed |

Table 2.1: Overview of possible attacks when parties collude

The Access Manager controls who is allowed to access what data. Therefore it can collude with a participant, say $A$, and grant $A$ access to all data. The Tweaker then re-keys data to $A$ and re-shuffles pseudonyms to their local versions for $A$. Whether or not $A$ can connect these local pseudonyms to real identities depends on the role of $A$. Doctors can make such connections, but researchers cannot. Such un-intended access by $A$ is registered by the Logger.

An overview of the attacks when the crucial parties (Key Server, Tweaker, Access Manager) collude, possibly with others, is given in Table 2.1. For any of the attacks discussed before it is possible to include the Storage Facility to get direct access to the data and circumvent the Logging and Access Management. Thus it is clear that at least the Key Server and the Tweaker must be completely independent and must run their cryptographic tasks and store their key material in HSMs.

### 2.6.2 Security assumptions

In this subsection we briefly discuss the cryptographic (number theoretic) assumptions that underlie PEP security. The basic PEP scheme in Section 2.1 uses the ElGamal encryption scheme [8] in the cyclic group $\mathbb{G} = \langle g \rangle$ of prime order $p$. That is, any element in $\mathbb{G}$ can be uniquely written as $n \cdot g$ with $n \in \{0, 1, \ldots, p-1\}$ or equivalently with $n$ an equivalence class modulo $p$. The latter classes are called the Galois field of order $p$ denoted by $\mathbb{F}_p$. The security of the ElGamal scheme depends on two cryptographic assumptions respectively called the *elliptic curve discrete logarithm (ECDL)* assumption — as we work

with elliptic curves — and the *computational Diffie-Hellman (CDH)* assumption in $\mathbb{G}$. We already encountered the DL problem, on which the DL assumption is based, in Section 2.1. It states: given $n \cdot g \in \mathbb{G}$ find $n \in \mathbb{F}_p$. The CDH assumption relies on a variation (and in fact an relaxation) of the DL problem. The CDH problem states: given $\alpha = n \cdot g, \beta = m \cdot g$ determine the value $(n \cdot m) \cdot g$. This problem is related to the security of the Diffie-Hellman key exchange problem [6], the first published public key encryption scheme.

In the context of the ElGamal scheme, if follows from the assumed hardness of the ECDL problem that one cannot derive the private key $x$ from the public key $y = x \cdot g$. Moreover, it follows from the assumed hardness of the CDH problem that one cannot determine the plaintext $M$ from an ElGamal encryption $\mathcal{EG}(r, M, y)$ without possession of the private key $x$. In practice however, it does not suffice that outsiders cannot determine $M$ from its ElGamal encryption but one additionally requires that outsiders cannot gather 'any' information on $M$ from its ElGamal encryption. This is formalised in *semantic security*: an outsider should not be able to determine whether two different ElGamal encryptions under the same public key $y$ contain the same message. That is: given $\mathcal{EG}(r_1, M, y)$ and $\mathcal{EG}(r_2, N, y)$ it should not be feasible to determine whether $M = N$. Semantic security of the ElGamal encryption scheme is related to a third mathematical problem in $\mathbb{G}$ called the *Decisional Diffie-Hellman* (DDH) problem with respect to $g$: given $g, m \cdot g, n \cdot g, r \cdot g$ all in $\mathbb{G}$, with random numbers $m, n, r \in \mathbb{F}_p$, determine if $n\dot{m} = r$. It is clear that the DDH problem is weaker than the CDH problem. One can easily show that the CDH problem is weaker than the ECDL problem in $\mathbb{G}$. Therefore we require a group $\mathbb{G}$ in which the DDH problem is hard in the context of PEP. This requirement is widely assumed to be met by all groups that are typically used in the context of ElGamal such as the multiplicative group of a finite field, the NIST elliptic curves [17] and the Brainpool curves [14].

It directly follows from the semantic security of the ElGamal encryption scheme that re-randomized versions of polymorphic pseudonyms, encrypted pseudonyms and encrypted data in the sense of Lemma 2.1.2 are not linkable to the original version. From the hardness of the elliptic curve Discrete Logarithm problem in $\mathbb{G}$ it also follows that a participant cannot cryptographically link a local pseudonym with the identity pid of the patient involved. Moreover, from the Decisional Diffie-Hellman problem in $\mathbb{G}$ it follows that local pseudonyms are not linkable over the participant domains. That is, it is not cryptographically feasible that two participants, say $C, D$, can assess whether local pseudonyms $\text{pid}_A@C$ and $\text{pid}_B@D$ correspond to the same person, *i.e.* whether $A = B$.

These properties hold under the assumption that the Tweaker and the Key Server are trusted parties that protect their cryptographic keys and do not deviate from the protocols described. The consequences of a deviating Tweaker are already sketched in Subsection 2.6.1.

As already mentioned in Subsection 2.3.1 we envision the usage of Hardware Security Modules (HSMs) to enforce that the Tweaker and Key Server do not deviate from their cryptographic tasks and responsibilities. The envisioned HSMs will not only manage the cryptographic keys involved in a non-exportable

fashion but will also enforce a usage policy. For instance, the Tweaker HSM will only allow the re-keying and re-shuffling operations from Section 2.1 and nothing else. Such enforcement is common practice in the smartcard industry, *e.g.* in EMV cards [10] used in the financial industry. Inside such card complete, albeit simple, applications are implemented using cryptographic keys stored in the card in a specific manner. Modern HSMs also support implementation of such applications.

In Subsection 2.5.2 we actually mimicked the usage of HSMs in software using pairing based cryptography. The pairing based context comprises of three groups $\mathbb{G}, \mathbb{H}$ and $\mathbb{K}$ in which we assumed that all three of the ECDL, CDH and DDH problems are intractable. This is assumed for the Barreto-Naehrig curves we mentioned in Subsection 2.5.2. Actually, these curves are assumed to be so-called Type 3 pairing friendly curves, *cf.* [11]. This means that in addition there are no efficiently computable homomorphisms between $\mathbb{G}$ and $\mathbb{H}$.

In the pairing based setting explained in Subsection 2.5.2, the Tweaker no longer has possession of the key factor $K_A$ as a number but only in the encrypted form as a group element $K_A \cdot h$. As the ECDL problem is assumed to be hard in $\mathbb{H}$, the Tweaker cannot (and in fact no party at all) determine $K_A$ from $K_A \cdot h$. This suggests that a Tweaker colluding with the participant $A$ is no longer able to derive the secret master private key $x$. This can be formally proven under the assumption of the hardness of the co-elliptic curve Discrete Logarithm (co-ECDL) problem. This problem resembles the regular ECDL problem but divided over both groups $\mathbb{G}$ and $\mathbb{H}$ as follows: given $g, n \cdot g \in \mathbb{G}$ and $h, n \cdot h \in \mathbb{H}$ with random $n \in \mathbb{F}_p$, determine $n$. It seems like a reasonable assumption that the co-ECDL problem is hard in Barreto-Naehrig curves [18].

Additionally, in the pairing based setting, the Tweaker is no longer able to perform transformations of pseudonyms between domains, as pictured in Diagram 2.13. The idea is that the Tweaker is no longer in possession of the pseudonym factors of the participants required to perform this domain transformation. To determine the pseudonym factor of a participant $B$ the Tweaker is required to calculate the discrete logarithm of $h_B$ with respect to $h$ which we assumed is intractable. Formally one can show that the ability to perform a domain transformation implies solving the DDH problem in $\mathbb{H}$ which we assumed is a hard problem. Therefore, no party is able to perform domain transformation in the pairing based variant. In a similar fashion one can show that the Tweaker in the pairing based variant is no longer able to transform an encrypted pseudonym for participant $A$ to an encrypted pseudonym for another participant $B$. We conclude that in the pairing based variant no party, including the Tweaker, is able to cryptographically conduct domain transformations.

We have indicated that in the pairing based setting the conversions from polymorphic to encrypted pseudonyms and from encrypted pseudonyms to pseudonyms are strict one-way functions. That is, it is not possible to convert (encrypted) pseudonyms to other (encrypted) pseudonyms. This suggests a robust setup.

### 2.6.3 Implementation

Currently, a prototype implementation of basic PEP functionality is under development. The software is written in C++ and uses RELIC[8] as the underlying crypto library — since it supports pairings. In this preliminary setting all parties (from Figure 1.1) are processes that run on the same machine. During further development more and more roles will be distributed to other machines and organisations. The PEP-software will become publicly available, as open source.

The following code snippet gives an impression. It implements the re-randomisation function $\mathcal{RR}$ from Lemma 2.1.2 (1). The triple of points $\langle b, c, y\rangle$ on the curve is an object of the class `ElgamalEncryption`. It is re-randomised via the formula (2.4), with randomisation parameter `z`.

```
ElgamalEncryption ElgamalEncryption::rerandomise() const
{
        ElgamalEncryption r;
        uint8_t randomness[32];

        MiscCrypto::randomBytes(randomness, 32);
        CurveScalar z(randomness, 32);

        r.b = b.add(CurvePoint::baseMult(z));
        r.c = c.add(y.mult(z));
        r.y = y;
        return r;
}
```

The following two design goals are intended to contribute to (wide) adoption of the PEP-framework.

1. Simple adaptation for software developers: minimum effort should be required to design software extensions of PEP or to modify existing software for PEP compatibility. Complex cryptography and protocol handling is abstrated away, so that adoption can be seamless and quick. Additionally, the PEP code is designed to be highly modular. Hence, developers may choose to replace certain functionality, *e.g.* I/O or cryptographic libraries, with custom implementations should they feel the need to.

2. Low memory footprint: in order to allow for adoption among an extensive range of (possibly embedded) devices, the amount of memory required is kept to a minimum.

Furthermore, the implementation includes bindings for several languages including Java, Python and C#.

Currently, the implementation of PEP is actively developed. Most of the low-level functionality has been implemented. Simple tests indicate that it performs well on ordinary consumer-grade hardware.

[8] See `github.com/relic-toolkit`

## 2.7 Conclusions and future work

We have described the basic ideas and application scenarios for Polymorphic Encryption and Pseudonymisation (PEP). The underlying mathematics is relatively easy, but the application protocols are complicated. Here is a list of things that we still wish to do.

- Interact with health care professionals to see which protocols best suit their mode of work.
- Interact with manufacturers of wearables and health care devices to work towards integration.
- Develop formal security proofs to substantiate the reasoning in Subsection 2.6.2.
- Analyse the relevant security protocols with tools like ProVerif[9].
- Extend the prototype implementation of the PEP framework, and provide open APIs to connect to it.

### Acknowledgements

Thanks to Simon Jacobs for producing the art work in Chapter 1.

[9] See `http://prosecco.gforge.inria.fr/personal/bblanche/proverif`.

# Bibliography


[1] G. Alpár. *Attribute-Based Identity Management*. PhD thesis, Radboud Univ. Nijmegen, 2015.

[2] G. Alpár and B. Jacobs. Credential design in attribute-based identity management. In R. Leenes and E. Kosta, editors, *Bridging distances in technology and regulation*, 3rd TILTing Perspectives Conference, pages 189–204, 2013.

[3] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000. Freely available via `www.credentica.com`.

[4] J. Camenisch and A. Lehmann. (un)linkable pseudonyms for governmental databases. In *Computer and Communications Security*, pages 1467–1479. ACM, 2015.

[5] J. Camenisch and E. van Herreweghen. Design and implementation of the Idemix anonymous credential system. In *CCS'02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 21–30. ACM, 2002.

[6] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, 22(6):644–654, 1976.

[7] J. Domingo-Ferrer, editor. *Inference Control in Statistical Databases*, number 2316 in Lect. Notes Comp. Sci. Springer, Berlin, 2002.

[8] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. on Information Theory*, 31(4):469–472, 1985.

[9] K. El Emam, E. Jonker, L. Arbuckle, and B. Malin. A systematic review of re-identification attacks on health data. *PLoS One*, 6(12):e28071, 2011.

[10] EMVCO. EMV integrated circuit card specifications for payment systems, book 2, security and key management. Version 4.3, see `emvco.com`, Nov. 2011.

[11] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journ. of Cryptology*, 23(2):224–280, 2010.

[12] D. Galindo and E. Verheul. Microdata sharing via pseudonymization. Technical report, Radboud Univ. Nijmegen, 2007. Joint UNECE/Eurostat work session on statistical data confidentiality, see `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.599.1265&rep=rep1&type=pdf`.

[13] J. Heurix and T. Neubauer. Privacy-preserving storage and access of medical data through pseudonymization and encryption. In S. Furnell, C. Lambrinoudakis, and G. Pernul, editors, *Trust, Privacy and Security in Digital Business*, number 6863 in Lect. Notes Comp. Sci., pages 186–197. Springer, Berlin, 2011.

[14] IETF. Request for comments 5639. Elliptic curve cryptography (ECC) Brainpool standard, curves and curve generation. see `ietf.org/rfc/rfc5639.txt`, March 2010.

[15] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *30th IEEE Symposium on Security and Privacy (S&P 2008)*, pages 111–125. IEEE, 2008.

[16] H. Nissenbaum. *Privacy in Context. Technology, Policy, and the Integrity of Social Life*. Stanford Univ. Press, 2009.

[17] NIST. Digital signature standard (dss). Federal information processing standards publication, FIPS PUB 186-4, July 2013.

[18] G. Pereira, M. Simplício, M. Naehrig, and P. Barreto. A family of implementation-friendly BN elliptic curves. *Journ. Systems and Software*, 84(4):1319–1326, 2011.

[19] N.P. Smart. Elliptic curve based protocols. In I.F. Blake, G. Seroussi, and N.P. Smart, editors, *Advances in Elliptic Curve Cryptography*, number 317 in LMS, pages 3–19. Cambridge Univ. Press, 2005.

[20] M. Veeningen, B. de Weger, and N. Zannone. Models and proofs of protocol security: A progress report. In A. Jøsang, P. Samarati, and M. Petrocchi, editors, *Computer Aided Verification*, number 7783 in Lect. Notes Comp. Sci., pages 145–160. Springer, Berlin, 2013.

[21] E. Verheul. Privacy protection in electronic education based on polymorphic pseudonymization. See `eprint.iacr.org/2015/1228`, 2015.