Master thesis

Computing Science
TRU/e Master in Cyber Security

Radboud University

Backup and Recovery of IRMA Credentials

Author:
Ivar Derksen
`ivarderksen@outlook.com`

University supervisor and first assessor:
Prof. Dr. Bart Jacobs
`bart@cs.ru.nl`

Second assessor:
Dr. Hanna Schraffenberger
`H.Schraffenberger@ru.nl`

External supervisor:
Timen Olthof, MSc
`timen@timenolthof.nl`

May 24, 2019

Abstract

This master thesis is a contribution to the IRMA project, a privacy-friendly identity management platform. We focus on the possibility to introduce recovery of IRMA credentials stored on a user's device.

We conclude that a backup and restore solution is needed for recovery. The backups should be properly encrypted to preserve the privacy and security guarantees of IRMA. The decryption key of the backups should be stored externally from the device in a safe way. Our research points out that the best solution for this is to use two-factor authentication where a part of the key is converted to a mnemonic phrase, using words that a user can physically write down, and another part of the key that is stored at a trusted party. Another finding is that in the restore process device revocation should be a mandatory element. When a backup is restored on a new device, the IRMA app on the old device should be invalidated. To realize this feature in the existing protocols, we introduce a device key. This is a symmetric key and because of this, the user's IRMA secret key can be modified without the need to change the electronic signatures of credentials in which the secret key is included.

Furthermore, we designed a proof-of-concept of recovery functionality in IRMA, including the design of protocols for the needed communication with other parties. For the encryption of backups we introduce the matryoshka encryption model with multiple encryption layers to enforce participation of the keyshare server in the recovery process.

Acknowledgements

I want to thank all family, friends and other people that somehow helped me with making and finishing this master thesis project. There are some people that I would particularly like to thank.

- Bart Jacobs for giving me feedback on this research and encouraging me to get the most out of my writing skills.
- Timen Olthof for his effort to supervise me.
- Koen van Ingen for helping me with all ins and outs of IRMA.
- Tim Vermeulen for the effort he did in helping me with time management and giving feedback on this thesis.
- All people in the IRMA team and in the Alliander teams for their input to this thesis.

Contents

1. Introduction

We prove who we are to online services every day. In contemporary online society, people visit online services like social media, web shops and internet banking multiple times a day. For all those websites user authentication is needed. Many organizations use passwords for this, but this authentication mechanism is reaching its limits. Florêncio & Herley [12] concluded that in 2007 a user had 25 passwords on average of which most were just a variation on another password. Counting variations as one password, a user only had 7 passwords on average. Since then even more online accounts were introduced and the password requirements became stricter and stricter. Therefore the problem is probably even higher today. Another disadvantage according to Florêncio & Herley [12] is that many people have to use additional tooling (pieces of paper, password reset features, password managers, etc.) to be able to remember the passwords.

Over the years multiple solutions have been conceived to solve the issue of passwords and identities. Allen [1] describes four categories for identity management systems:

- **a centralized model** is a model in which there is one authority that is responsible for dealing with identities in a certain area. In this approach you always rely on the judgment of the centralized authority whether some identity claim is right or not. Responsibilities can be delegated in a hierarchy to make it easier. A good example are domain names. The centralized authority for domain names, ICANN, manages which parties are responsible for all top level domains. For example it states that SIDN is responsible for the top level domain `.nl`. SIDN is then responsible for managing who owns all second level domains (for example `ru.nl`), and so on. At every level is defined who is responsible and in order to trust a domain you have to trust all parties in the domain trail. In the end everything falls back on the root trust that ICANN has and it therefore has all power.

- **a federated model** where a system does not do user authentication itself, but trusts user authentication done by another system. A good example of this is the eduroam university wireless network in which local access is also granted to people that have credentials for other universities. In the centralized model the user administration of a particular university would be used as authority. Then people from other universities are never able to use the wireless network. A federated system also accepts statements from trusted parties that a particular user should be allowed. The system then trusts that another party authenticated the user and builds upon its judgment. An advantage is that the number of different credentials in this system is much lower, because you only need credentials at one party to be able to log in at all other parties. A disadvantage is that for the security, organizations rely on systems of other parties.

- **a user-centric identity model** where a single credential provider is responsible for what about a user is shared with which party. The idea is that the user only authenticates to a credential provider and this party manages all identifiers of the user. Via the credential provider the user can then authenticate to other service providers. For each service provider the user is able to select which information is shared by the credential provider. The credential provider should be trusted to only share the selected information and not more. Good examples of this are Google Sign-In or Facebook Connect.

  At a service provider a user can choose to authenticate using credential providers like Google or Facebook and then the user gives that credential provider permission to share certain information with the service provider. An advantage of this is that only one login credential is needed for a lot of services and a user has control over what is accessed by whom via the credential provider. A disadvantage is that the credential provider gets a lot of power. Credential providers can set all kinds of conditions that service providers have to fulfill to use the authentication service. Service providers are now dependent on the cooperation of the credential providers. If a credential provider refuses to give access at some point or they supply wrong information, the particular service provider has a problem. From a user's perspective there are large privacy concerns when all data is stored at one party.

- **a self-sovereign identity model** where the user's personal data is not only organized around a central point, but actually is in the control of the user himself. As Allen [1] describes it, the user should "rule" over his own identity. Instead of data being managed by a trusted party, the data should be managed by the user. In this way, the user is self-sovereign over his data and determines himself what happens with the data.

1.1. IRMA Explained

IRMA [3] is an identity platform that is based on the concept of a self-sovereign identity. The main concept of the platform is that the identity of a person is splitted into multiple attributes containing only one element of the identity. This can be a general property like being a student or being over 18, but it can be very detailed like a social security number. Users have to collect attributes at issuers. Issuers are parties that can do claims about the identity of the user, like the government. Users have to show to an issuer the relevant parts of his identity and then that issuer grants validated attributes to the user. When the user visits some relying party that wants to know a certain property of the user, it can show the relevant attributes only instead of showing his full identity. IRMA can also be used for signing messages using selected attributes, so people do not have to sign with their own unique identity. They can also sign with their attribute `employee of company X` without leaking other identifying information.

Every user has an IRMA token (for example an app on a smartphone) to store his attributes in. The user always has to explicitly give consent to the token (for example with a PIN code), so the user decides what to share with whom. The token is in control of the user, so this also holds for the attributes.

The technique of IRMA is based on the Idemix anonymous credential system [5]. A detailed explanation of the working of IRMA can be found in chapter 2.

1.2. Usability

To improve the user experience of online user authentication it is not only important that an independent overarching identity platform is introduced as an alternative for the many authentication services currently available on the market. The usability of the identity platform itself, like IRMA, should also be good enough to be a good alternative for users.

Alliander ran into some practical issues related to the user experience of IRMA. The biggest practical limitation they mentioned is the process of switching to a new device. Now, when a phone is lost, broken or replaced and the user gets a new phone, all attributes the user collected are lost too. The user should then fully set-up the IRMA app again on its new phone and collect all attributes again. Especially when a user collected many attributes or when he has to physically visit a location to get certain attributes, this can be much effort. In this research we therefore focus on the possibility of introducing recovery functionality in IRMA. The other practical limitations Alliander experienced are mentioned in future work (chapter 9).

1.3. Research Overview

Recovering attributes is not a trivial feature for IRMA, because attributes contain privacy sensitive information and secret key material that cannot be stored somewhere in plain. On the other hand the solution must be usable for any user of the app, so it should also be user-friendly. This means we have to find a solution that meets both sides of the spectrum. This leads to the following research question.

> What are the possible strategies for recovering IRMA attributes on a new device and what are the implications of those strategies?

As mentioned before, we first discuss some technical background about IRMA in chapter 2. Besides a general introduction, also details about the IRMA app and the protocols are explained here. The rest of this research is built on this knowledge.

From then our research is built up in three parts. At first we introduce design criteria for a proper recovery solution for IRMA. In chapter 3 we discuss an overview of design principles in the field of self-sovereign identity platforms known from law and literature. In chapter 4 we apply these principles on IRMA and based on that we introduce the design criteria for recovery functionality within IRMA.

In the second phase we present a design for a proper recovery solution that sufficiently meets all criteria we defined in the first phase and we consider several technical solutions to fill in the key elements of this design. In chapter 5 we first examine how other self-sovereign identity platforms solved the issue of recovery. Based on the ideas we got from other platforms and the design criteria we composed in chapter 4, we describe in chapter 6 what strategy suits the best for recovery in IRMA. In this chapter we also consider what technical solution suits best to implement this strategy.

Finally, we designed a proof-of-concept to show how our recovery design can be included in the implementation of IRMA. In chapter 7 we give a technical explanation of the details of this proof-of-concept.

2. Technical Background

Recovery as a functionality in IRMA cannot be seen separately from all the other IRMA functionalities. Therefore we first explain the working of the current IRMA protocols and the communication with other parties needed for this.

IRMA works according to the principle that issuance of personal data is separated from the usage of personal data. The kind of procedures needed for IRMA is roughly the same as the current models in passports, identity cards and driving licenses. In here the owner of a document first gets issued an identity document from a governmental institution like a municipality. When he receives the document he can use the document for any purpose the owner likes without the issuing institution being involved, but the guarantees on the validity of the document do still hold.

The same analogy holds in IRMA. The user gets issued IRMA credentials from a party claiming something about that user. For example a bank can declare a certain bank account is owned by the user, the municipality can declare a user lives on a certain address, etc. An issuer declares this by signing the information and including an expiry date about how long the information is valid. This is done in an issuing session. Simply said, a unit of information that is issued combined with the issuer's signature on this information is called a credential. A more detailed explanation of what a credential exactly is can be found in section 2.1. The user stores the credentials in his IRMA app and can then disclose attributes from these credentials or sign messages with certain attributes. This is done in a disclosure or signing session. The issuer of the credential is not involved in usage of credentials issued by him.

We discuss the outline of the protocols of disclosure and issuance in section 2.2. Disclosure and signing work practically the same, so we only discuss the disclosing session. The exact outline of storage in the IRMA app can be found in section 2.3. For recovery making backups of the user data might be necessary and for that it is important to know how storage in IRMA is organized. Finally, we describe briefly in section 2.4 what guarantees IRMA offers due to the setup we explained in the sections before that. Recovery should not break these key principles.

2.1. IRMA Credential Build-up

Vullers & Alpár [32] describe a credential as a "cryptographic container of attributes". The attributes contain the actual information. In figure 2.1 the IRMA credential outline is illustrated. In all IRMA credentials the first attribute is the *secret key*. The second attribute contains all the metadata about the credential such as what type of credential it is, the issuance date and the expiry date. Upwards from the third attribute the included attributes depend on the type of credential. For example, Privacy by Design Foundation's email credential (`pbdf.pbdf.email`) only contains one additional attribute: an email address of the user, whereas the agelimit credential from the municipality of Nijmegen (`pbdf.nijmegen.ageLimits`) contains five additional attributes: `over12`, `over16`, `over18`, `over21` and `over65`. An attribute can contain information in raw bytes of any length. Currently the IRMA app only supports multi-language text strings, but in theory attributes can be of any type.

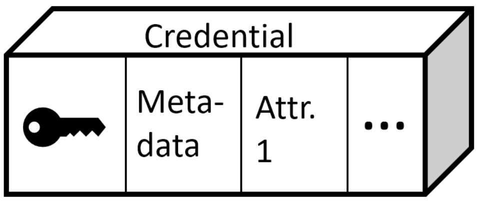
A credential is signed by the issuer. Not the credential itself is signed, but it is done using a signature on a zero-knowledge proof commitment of all attributes in the credential. IRMA uses the Idemix Anonymous Credential System [5] for this. The signature is placed on a commitment, because when all attributes of a credential would be included directly in the signature all attributes always have to be disclosed in order to check the signature. By using a commitment there is no need to show all attributes in a credential. A user can prove certain attributes are included in the commitment without having to show the other attributes in the credential. In this way IRMA achieves *selective disclosure*.

A verifier can check the validity of a signature on a zero-knowledge proof using the public key of the issuer of the credential. IRMA knows *scheme managers* that are trusted with the task of checking the validity of public keys. These scheme managers are the root of trust within IRMA. They determine what credentials are valid according to them, how that credential looks like. This is needed in order to create a single truth between all parties that use IRMA. IRMA allows multiple scheme managers to be used. The scheme manager is also responsible for measures to prevent abuse of credentials in their scheme. For example, a scheme manger can specify the keyshare server that should be used. The explanation of the keyshare server can be found in section 2.2.

Secret key The most important attribute in all credentials is the user's secret key. As mentioned above this always is the first attribute in every credential. The key is 256 bits in size and is generated during initialization of the IRMA app, secretly and with proper randomness. There are no other requirements on the generation process. The secret key is only known by the user. Its value is never disclosed to any party. Even during issuance, the issuers are not able to learn the user's secret key. The user includes the secret key in the commitment of the zero knowledge proof, producing the credential's signature, without sharing it. The issuer places a so called blind signature on the key.

The presence of the secret key has two reasons. On the one hand it enforces control of the user. A user can only disclose attributes from a certain credential when it knows the value of all attributes in the credential; even when not all attributes are disclosed. Since the secret key is one of the attributes, the secret key is needed in order to do a disclosure. Given that only the user knows his own secret key, we have the guarantee that the user participated in the disclosure process. The IRMA app enforces that the secret key stays protected. On the other hand, the secret key is important when attributes of multiple credentials are combined. A proof can disclose attributes from multiple valid credentials at the same time, but this does not necessarily mean that those credentials belong to the same user. However, for a disclosure it is essential to know that all credentials belong to each other. Otherwise a user can compose a fictive identity of his choice by combining credentials of several users. Therefore, when multiple credentials are disclosed at the same time, the verifier additionally checks that the secret key is the same in every credential included in the disclosure. This gives the guarantee that all credentials concern the same user.

Figure 2.1.: Outline of an IRMA credential



2.2. IRMA Protocols

The actors in IRMA's procotols are the **user** having an instance of the IRMA **app**, the **issuer** (in case of issuance) or the **verifier** (in case of disclosure) both having an instance of the IRMA **API server** and, finally, the IRMA **keyshare server**. A verifier in most cases is a **relying party** that uses IRMA to authenticate at his services.

The app handles the communication of the user, the IRMA API server handles the communication of the issuer/verifier and the keyshare server is involved to guarantee the user actually entered his PIN code. Only the user interacts with the keyshare server. The keyshare server participates in the proof if and only if the user authenticated using the right PIN. Via such remote checking IRMA guarantees the PIN code cannot be brute-forced using an offline attack on the user's device. The keyshare server has the ability to take measures against brute-force PIN attacks.

The protocol descriptions below are based on Privacy By Design's issuance and disclosure protocol explanation [14] and Privacy By Design's keyshare protocol explanation [15].

2.2.1. Issuing Session

An issuing session starts with an issuer that sends an issuance requests to his API server. In such a request he sends what credentials are going to be issued, what the values of the attributes in those credentials should be and optionally he specifies what attributes the user has to disclose as a condition to get the credentials. This is needed when credentials are based on other credentials and then it should be checked whether the user actually has those credentials. If this request is valid, the issuer gets back a session token.

The issuer sends this session token and the URL of his API server to the user's app such that the user can retrieve his credentials at the API server. In most cases this happens by the user scanning a QR code containing the session token. The user's app sends the token to the API server and gets back a list of the credentials being issued, a nonce to guarantee freshness and optionally the list of attributes having to be disclosed. The user's app shows this information to the user and asks for permission to store the new credential.

When this permission is given a proof is being constructed. This happens via a keyshare session which we will discuss in Section 2.2.3. After the keyshare session a commitment on the earlier received nonce is sent to the API server together with a response of both the user (s) and the keyshare server (s_k). These are the responses on the zero-knowledge proof. A zero-knowledge proof is also needed in issuance since the issuer is not allowed to learn the user's secret key and this key is included in the credential. The response of the keyshare server (s_k) is also signed by the keyshare server to make sure that a user is not able to circumvent the keyshare server. The API server checks whether the response, the commitment and the nonce are right for that session and whether the signature of the keyshare server is correct. If this is the case, it sends the new Idemix credential to the user's app.

A visualization of this protocol can be found in figure 2.2.

2.2.2. Disclosing Session

The disclosure protocol is very similar to the issuance protocol. In the beginning a relying party (lets say a web shop) wants to check some attributes of a user and therefore becomes a verifier. He sends a disclosure request to his API server and then the same process starts as during issuance. A small side note: the verifier does not have to run his own instance of the API server, he can also use one he trusts as a broker.

The only difference compared with issuance is that when the user's app sends the IRMA proof to the API server (at the step DisclosureProof), the app already combines the proofs of the keyshare server and the user. The verifier does not have to check the signature of the keyshare server, since he can trust the issuer already checked that and he has to trust the issuer anyway since that is the party that declares the disclosed attributes are correct. Combining the proof responses beforehand guarantees that the verifier cannot retroactively track down the user by forcing the keyshare server to disclosure for which user it did a particular proof. This is especially important when disclosing non-identifying attributes.

Instead of sending credentials to the user in issuance, the API server sends during disclosure the disclosed attributes to the verifier. The verifier knows this information is correct since the API server checked the validity of the proof. In case of a signature request, the signature is sent to the verifier along with the disclosed attributes.

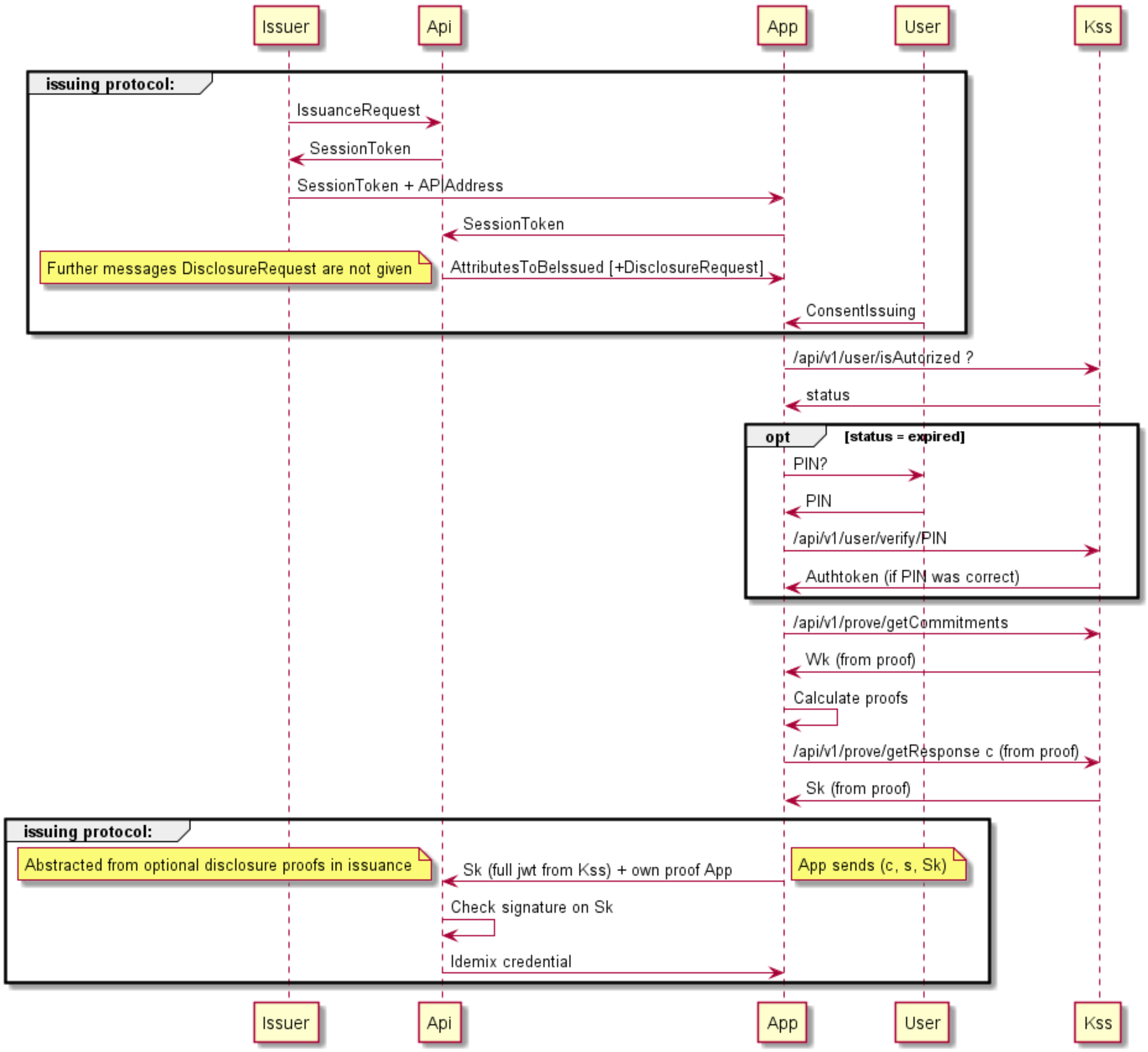
A visualization of this protocol can be found in figure 2.3.

2.2.3. Keyshare Session

The keyshare session is the session between the user and the keyshare server that has to be done to generate a multi-party proof of knowledge of the secret key. When a keyshare server is used, the secret key consists of a user's key part and a keyshare that is stored at the keyshare server. These two parts have to be combined. A keyshare session is the same for both issuance and disclosure. At first the user has to authenticate at the keyshare server. If the user was not already authenticated, the user's app has to request the user to enter his PIN. When the user entered the PIN, the app can let the PIN be checked by the keyshare server. If it was correct the keyshare server sends an authentication token that can be used to start a session to generate a proof.

At first the user's app asks the keyshare server to commit to a random number it generated for the proof by calling `getCommitments`. The keyshare server also generates a random number and it calculates a commitment on both his and the keyshare server's randomness, the data in the credential and the initial nonce for freshness (sent by the issuer's or verifier's API server). It sends this commitment to the keyshare server such that both the user and the keyshare server can calculate their responses based on respectively the user secrey

Figure 2.2.: Flow chart of an IRMA issuing session


Issuer
Api
App
User
Kss
issuing protocol:
IssuanceRequest
SessionToken
SessionToken + APIAddress
SessionToken
Further messages DisclosureRequest are not given
AttributesToBeIssued [+DisclosureRequest]
ConsentIssuing
/api/v1/user/isAutorized ?
status
opt
[status = expired]
PIN?
PIN
/api/v1/user/verify/PIN
Authtoken (if PIN was correct)
/api/v1/prove/getCommitments
Wk (from proof)
Calculate proofs
/api/v1/prove/getResponse c (from proof)
Sk (from proof)
issuing protocol:
Abstracted from optional disclosure proofs in issuance
Sk (full jwt from Kss) + own proof App
App sends (c, s, Sk)
Check signature on Sk
Idemix credential
Issuer
Api
App
User
Kss

The flow chart only covers the flow of messages of the protocols within IRMA. The issuance procedure as a whole may include more steps, like proper authentication of the user by the issuer.

Figure 2.3.: Flow chart of an IRMA disclosing session

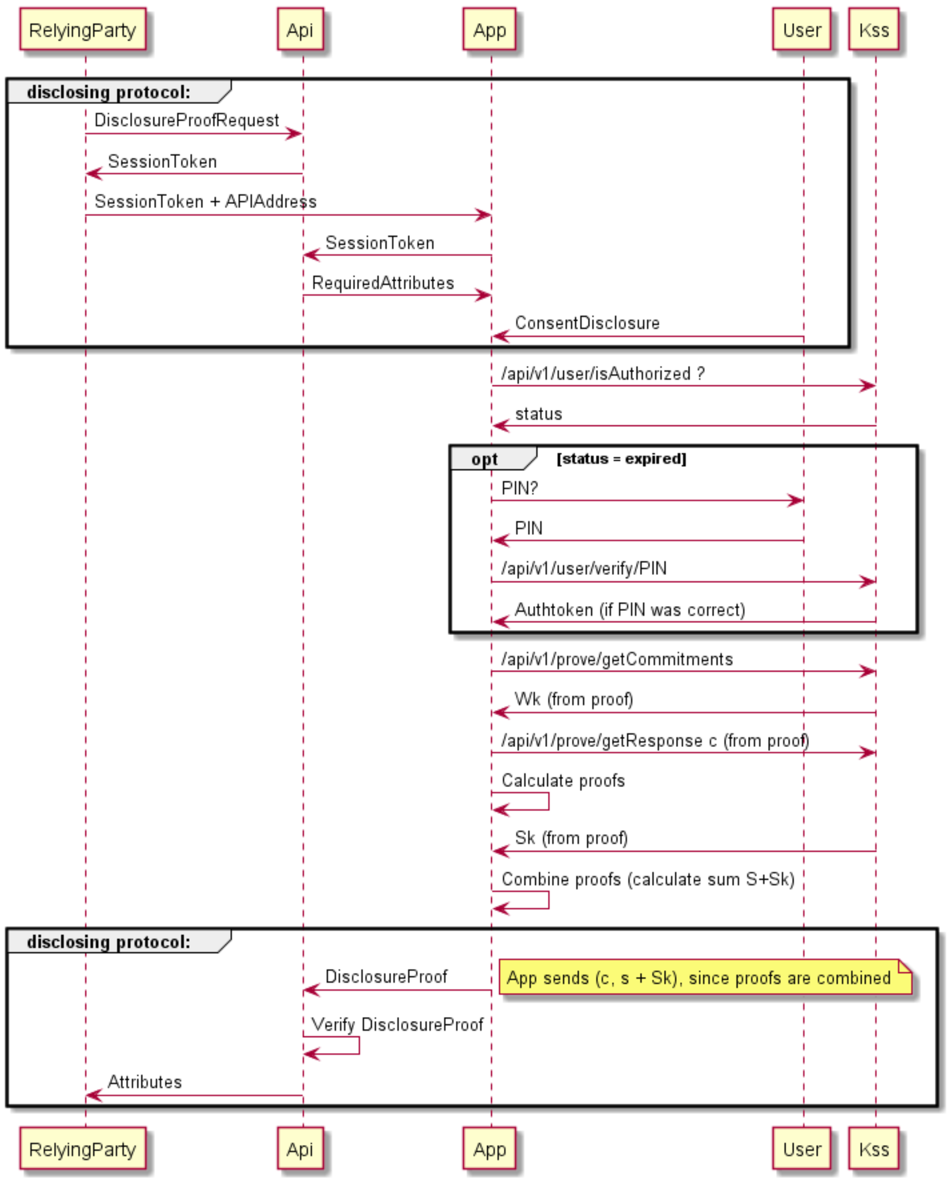


Figure 2.4.: IRMA app example file structure (Android)

```
/data/data/org.irmacard.cardemu
├── cache
├── code_cache
├── files
├── lib
├── lib-main
├── shared_prefs
└── v2
    ├── irma_configuration
    ├── sigs
    │   └── 0b011fef96088f33
    ├── attrs
    ├── kss
    ├── logs
    ├── paillier
    ├── preferences
    ├── sk
    └── updates
```

key and the keyshare. These two responses together form the actual proof. As mentioned earlier, only in case of disclosure/signing these responses are combined.

2.3. Storage in IRMA App

In IRMA the user himself is in control of his credentials, so the IRMA app is responsible for storing the information related to this. All data is stored into internal app storage. The device's operating system makes sure that the data cannot be accessed by other applications or when connecting the device to a PC. Apps work in a sandboxed environment [9], so on non-rooted devices the IRMA app only can access the IRMA app's internal storage.

In figure 2.4 we displayed the internal storage of the IRMA app. All user data is stored in the folder `v2`. Below we will describe per file what is stored in it.

- **irma_configuration**:
  In this directory the information is stored about what issuers exist, what credentials they issue and what the public keys of those issuers are. This data is not user specific and will be the same for all users. Although it is publicly known information that is stored, it is included to make sure the integrity of the public keys can be guaranteed. You could compare this configuration to the root certificate chain of TLS certificates. It is a trusted basis the app can rely on when checking the integrity of newly downloaded information.

- **sigs**:
  In this directory the signatures of issuers are stored of all credentials. Each credential

has its own file in this directory. The file name is an identifier to the credential. A particular signature is needed when disclosing attributes from the credential to which it belongs.

- **attrs**:
  In this file all attributes are stored. The file contains a JSON array which contains JSON objects for every credential a user possesses. The JSON object of a credential has one element `Ints` which is an array with big integer representations of all attributes. Only the first attribute of each credential, the secret key, is not included. This is stored in `sk`. The first integer that is recored always is the metadata attribute. The first bit of this integer indicates the version of the credential. Therefore, always a bit shift is needed to derive the actual metadata. For the other attributes the big integer can just be translated to bytes and converted to ASCII to read the content.

- **kss**:
  This file contains the information of all keyshare registrations a user has. For example the URL of the keyshare server is stored, the username the user got, etc.

- **logs**:
  This file contains the log files of the IRMA app. All actions (disclosure, issuance, signing) performed with the app are logged in this file.

- **paillier**:
  This file contains the paillier keys of the user. These keys were used in a keyshare session to encrypt the information send to the keyshare server such that the keyshare server can never find out what credentials a user discloses to some party. The Paillier encryption system is used for this, since the cryptographic operations the keyshare server has to do can be done without the need to decrypt the data. In newer versions of IRMA another solution is used for this and then this file is not used anymore.

- **preferences**:
  In this file the app preferences of the user are stored. Currently this only concerns the option whether sending crash reports is enabled, but in the future more preferences can be added to this file like preferred app language, etc.

- **sk**:
  In this file the secret key of the user is stored. This is a 256 bit key. This value is used in the first attribute of all credentials. For credentials belonging to scheme managers using a keyshare server, this key is half of the first attribute. The other half is stored at the keyshare server. When no keyshare server is used, this first attribute of the credential fully matches with this secret key.

- **updates**: This file contains log information about updates of the app. Logging in this file takes place when data migrations in this `v2` folder are made. When there is an integrity failure in `irma_configuration` and a fallback has to be done on the apps default configuration, this is also logged here.

2.4. IRMA Security and Privacy Guarantees

Due to the discussed build-up of a credential, IRMA can offer multiple security and privacy guarantees on the user's data. Here we discuss the most important ones based on the analysis of Vullers & Alpár [32]:

- **Selective disclosure**: the user is only required to disclose the attributes that are needed in that particular scenario.
- **Authenticity and integrity**: credentials that can be proved valid are legitimately issued by the credential's issuer and belong to the particular user. There is no way to produce or modify a valid credential without cooperation of the issuer.
- **Non-transferability**: credentials cannot be transferred to other users, even not when the legitimate user agrees with this. In IRMA this is realized using the secret key as mentioned before. This key is well protected to prevent that a key can be copied and used at multiple places by different users.
- **Issuer unlinkability**: an issuer is not able to link usage of his credential in a verification process to a particular issuing session. In other words, when a credential is used, issuers cannot find out to which user it belongs. It is important for the user's privacy that profiling of credential usage is impossible.
- **Multi-show unlinkability**: when a credential is showed to a verifier multiple times, the verifier is not able to link two IRMA sessions to each other. This guarantee can only be given when the disclosed attributes are not uniquely identifiable to the user. Otherwise the information itself can be used to link the two sessions.

3. Overview of Design Principles for Self-Sovereign Identity Platforms

In this chapter we give an overview of some visions on principles for self-sovereign identity platforms. These visions combine important ideas from researchers in the field of identity management. We use these visions as a foundation to determine what the requirements should be for a recovery solution in IRMA. This chapter gives background to Chapter 4 in which we describe the requirements we selected for the design of a recovery solution in IRMA. Therefore we solely discuss relevant visions here as such. The reflection is done in Chapter 4.

In Section 3.1 we discuss Allen's Self-Sovereign Identity Principles [1] and Cameron's Laws of Identity [6]. Besides principles, solutions also have to deal with laws around privacy, data protection and identity management. The requirements that arise from this source we discuss in Section 3.2.

3.1. Self-Sovereign Identity Principles

In Chapter 1 we introduced the concept of Self-Sovereign Identity from Allen [1]. Allen also formulated ten principles that characterize self-sovereign identity platforms [1]. These principles are an extension of Cameron's Laws of Identity [6]. The self-sovereign identity principles take into account some additional perspectives that are not included in Cameron's laws.

Because of the partial overlap between the theories of Allen and Cameron, we will discuss Allen's self-sovereign identity principles and then link those principles to Cameron's laws when there is overlap. Views that are only covered by Cameron and not directly by Allen we will discuss separately at the end of this section.

1. **Existence:** Users in an identity platform should correspond with a person in real life. This link is important for the authenticity of claims made using an identity platform. Important issues related to this principle are for example that the information about an identity must be correct, that the integrity of claims should be guaranteed and that there should be authentication checks linking the user using the platform to the actual person the claims are about.

   This involves the law 'Human Interaction' from the Laws of Identity [6] saying that the channel between the identity platform and the actual user should have strong authentication involved making sure that digital identities cannot be used by people other than the actual user owning the identity.

2. **Control:** Users should have the ultimate authority over their own identity. Another party should not be able to control personal data of the user, because this creates

the risk that this party can access, use or update information without the user's permission. Worst case an attacker might be able to impersonate the user. A user should determine what identity to use, what information is released and must agree with the purpose it releases information for.

This requirement is related to the laws [6] 'User Control and Consent', 'Directed Identity' and 'Consistent Experience Across Contexts'. 'Directed Identity' adds to user control that it should not be possible to get information about an identity without the initiative coming from the user. 'Consistent Experience Across Contexts' requires that identities should ideally be labeled with their context or at least some difference between contexts should be noticeable to make it easy for the user to see what kind of identity properties are disclosed at some moment. For example, when a user is asked to give his email address, clear labels like 'personal' or 'work' should be displayed in order to make sure a user does not disclose the wrong identity.

Having control does not only mean that the user decides what happens with his own identity. It also means that the user must understand the consequences of decisions he has to make and what he has to do to keep in control. If this is not clear, the fact that the user is in control does not add much value since there might be ways to fool the user and circumvent all technical measures easily. Therefore it is also important that the user must know how to keep his identity safe and secure.

Jøsang *et al.* [17] describe usability principles to make sure users understand the reason why certain security measures are taken and why certain actions are necessary. A security action is an activity that people have to do in order to deal with security measures. Sometimes users also have to make security conclusions. A security conclusion is something a user has to check in order to conclude there are no security issues at that moment. You can think of checking the green padlock in your browser. The principles read as follows:

a) Users must understand which security actions they have to perform and must understand the security conclusion that is required for making an informed decision. This means that users must understand what is required of them to support a secure transaction.

b) Users must have sufficient knowledge and the practical ability to make the correct security action. If after an action a security conclusion is needed, the system must provide the user with sufficient information for deriving the security conclusion. This means that it must be logically possible to derive the security conclusion from the information provided.

c) The mental and physical load of a security action and/or a security conclusion must be acceptable.

d) The mental and physical load of making repeated security actions and/or drawing repeated security conclusions for any practical number of transactions must be acceptable.

3. **Access:** All information and all claims in an identity platform concerning the user should be accessible by the user to be able to have an overview of what happened. It should not be possible for certain parties to limit a user's access to systems or data.

4. **Transparency:** The platforms used for identity management should be open. This means it must be clear how they work and how they are managed. The systems should be open-source and well-known algorithms should be used in it. In the Laws of Identity [6] the law 'Justifiable Parties' adds to this that the parties involved in the identity platform should have a justifiable reason to be involved. For example, users must be sure that certain parties are not involved to collect data or to sell advertisements using that data. According to Cameron one should be careful to involve parties that might have stakes other than managing identities in order to maintain the user's trust.

5. **Persistence:** Independent of the technique used to implement the identity platform, properties and attributes of an identity should be usable as long as the user wants and as long as it is valid. When credentials expire there should be ways to regain the identities for users when they are still valid.

6. **Portability:** Users should not be forced to use services of specific third parties. This creates dependence for users and puts those third parties in a power relation with the user. This is related to pluralism of technologies from the law 'Pluralism of Operators and Technologies' in the Laws of Identity [6]. According to Cameron [6] pluralism of technologies is important because it is very likely that at some moment scenarios come up for which the techniques of the particular platform are not practical. Then there should be a way to be flexible and adapt it or to use other techniques.

7. **Interoperability:** The identity platform should be widely usable. Any party must be able to use it and it should support enough different use cases to make it attractive to use. This is related to pluralism of operators from the law 'Pluralism of Operators and Technologies' in the Laws of Identity [6]. According to Cameron [6] users work in different contexts (governmental affairs, work, personal, etc.) and it should be possible to use the platform in all contexts without depending on another. For example you do not want to be forced to use information from your passport (governmental context) at your work. A user might want to use separate identifiers in order to separate personal and work affairs.

8. **Consent:** Users have to explicitly agree with the fact that parts of their identity are used by other parties and, for signing messages, a user should be involved to make sure signing is done explicitly. This requirement is also related to the Law of Identity [6] 'User Control and Consent'.

9. **Minimalization:** When disclosing data or signing messages, only the minimum amount of data needed for the purpose of the request should be shared with the relying party. This is related to the Law of Identity [6] 'Minimal Disclosure for a Constrained Use'. Cameron [6] adds to this that already at disclosure of information a reliant party should declare for what purpose information is needed. This means that even after a relying party received the data it is obliged to only use it for the purpose the user disclosed it for.

10. **Protection:** The system must protect the freedoms and rights of all users. It must for example make sure the data is securely stored and the privacy of users is guaranteed.

Allen deliberately keeps this purpose abstract and does not mention specific freedoms and rights. He does this, because he thinks a platform should be censorship resistant and force-resilient. We think the underlying idea is that all freedoms and rights needed to realize this now and in the future might be important to realize these goals. The Laws of Identity [6] do not specify such an extra lock on the door.

One part of Cameron's law [6] 'Justifiable Parties' is not fully referred to in one of the self-sovereign identity principles. For completeness we also mention this. According to Cameron disclosure of information can only be done to parties for which this information is necessary and is justifiable in the relation the relying party has with the user. We discussed this law under the principle 'Transparency' from Allen [1], since we think the most important aspect of this law is that it must be clear for users who gets what data for which purpose. To some extend this law is also covered by the principle 'Minimalization' from Allen [1], because the law also states only the minimum amount of data is allowed to be disclosed for specific purposes. Cameron's law however is more strict in this. In his opinion users must at least be convinced with a policy statement why a relying party asks some information. In more extreme cases it discusses the possibility to make it impossible for certain parties to ask specific information in the first place. For example you could think of enforcing that only the government can ask for the national identification number (BSN in the Netherlands). This kind of enforced minimalization is not part of Allen's Self-Sovereign Identity Principles [1]. This would limit the user's self-sovereignty.

3.2. Legal Requirements

As legal background we discuss the eIDAS regulation [29] concerning electronic identification, authentication and trust services, the General Data Protection Regulation [31] and the ePrivacy directive [27] and the coming ePrivacy regulation [28]. Since these regulations and directives are quite wide-ranging, we will only discuss the parts relevant for recovery. This legal overview is made to find the major attention points that can be important to take into account when designing a solution for recovery in IRMA. This cannot be used as a full legal analysis and we also did not take into account the relevant case law.

3.2.1. eIDAS Regulation

The eIDAS regulation [29] specifies a common foundation for secure electronic interaction between citizens, businesses and public authorities. It includes both requirements for electronic authentication and electronic signatures. This regulation can have consequences for the design of identity management platforms supported by the government. The regulation enforces the government of EU member states to aim for a technology neutral approach. This means that as long as an identity management platform complies with the regulation, other EU member states should allow it to be used too at their services. The obligation for EU member states to support the platform is not only a matter of compliance of the platform. Article 7 and 9 of eIDAS [29] specify some other conditions, from which the most important one is that a member state must formally notify other member states that a platform should be supported. Therefore it is not mandatory for a identity management platform to comply with eIDAS. When a member state never makes a formal notification, eIDAS has no legal consequences for a platform. However, when aiming for government support having eIDAS compliance as option is favourable.

The requirements of eIDAS differ for the desired identity assurance level (low, substantial or high). The assurance level indicates the degree of certainty certain information belongs to the user and is determined by the weakest factor in a claim. So when information is combined, the combined assurance level is equal to the lowest assurance level in the set. The assurance levels form a hierarchy, so for reaching assurance level substantial you also need to comply with the requirements from assurance level low. This means that when assurance level high is achieved, they can also be used at level low since they also meet the lower conditions. As an eIDAS compliant platform you ideally want to support all assurance levels in order to be able to support all possible use cases.

In the eIDAS regulation [29] requirements are included that are important in the perspective of recovery and making backups. In this section we only focus on the parts from eIDAS relevant for recovery of IRMA credentials.

Article 24 paragraph 2g specifies that the services must take measures to prevent forgery and theft of a user's identity and the corresponding data. Also, article 26 specifies that electronic signatures should be uniquely linkable to the signatory. Since IRMA also supports electronic signatures, this leads to the requirement that credentials should be protected against usage by other persons. For a recovery process this means that this process should also be protected against usage by other persons. In IRMA credentials are already protected with a PIN to protect against this. However, only relying on this would mean that a user has to completely revoke its credentials when he thinks his PIN is compromised.

One of the commission implementing regulation [30] adds what are the minimal technical specifications and procedures for "electronic identification means". The definition of 'means' is written having physical smart cards in mind. However, the law abstracts from specific techniques which makes this definition hard to read. An electronic identification mean is defined as the unit that contains the personal data and what is used for authenticating at online services. IRMA can also hold credentials issued by non-governmental parties. Concerning the issuer role, this regulation is only applicable to public authorities and legal entities appointed by national law. In IRMA the smallest unit that contains verified personal data is an IRMA credential. This is also what is used to prove your identity, since the credentials are the units that are signed by issuers. An IRMA credential therefore is the electronic identification means in our opinion. In terms of eIDAS IRMA would then be the electronic identification scheme. An electronic identification scheme is defined as a "system for electronic identification under which electronic identification means are issued to natural or legal persons".

The technical requirements from eIDAS's implementing regulation which are relevant for recovery are:

- An electronic identification mean should protect against duplication and tampering (paragraph 2.2.1 of the regulation).
- In case of suspension, revocation and reactivation of an electronic identification means (paragraph 2.2.3 of the regulation) three criteria should be met:
    1. It should be possible to suspend and/or revoke a means timely and effectively

2. It should not be possible to perform suspension, revocation and/or reactivation unauthorized. Only the owner of the means or an authorized institution must be able to do this.
3. Reactivation is only allowed when the same assurance requirements can be guaranteed after reactivation as before the suspension.

- In case of renewal or replacement (paragraph 2.2.4 of the regulation) the same requirements should be met as at issuance or the process must be based on (another) valid electronic identification means of the same assurance level or higher. In case the assurance level should be high, then the data of the electronic identification means should also be verified with an authoritative source. For example, when a credential of the municipality is renewed or replaced, a user can show the old credential to the municipality and getting a new one if everything matches. When the old credential is not there anymore or assurance level high is needed, then the municipality is obliged to also check the governmental personal records database whether the data on the mean corresponds to the actual data.
- If for the authentication mechanism (paragraph 2.3.1 of the regulation) personal data is stored, the information should be protected against data loss and data compromise, including offline analysis. Furthermore it must be impossible to perform brute-force, replay or man-in-the-middle attacks and it should not be possible to eavesdrop communication. Finally, authentication protocols should always be based on dynamic authentication (challenge-response, one-time pad, etc.).
- All media that contain personal data, cryptographic keys or other sensitive information should be stored, transported and removed securely (paragraph 2.4.6 of the regulation).

In conclusion, it is not mandatory for IRMA to comply with this regulation, but doing everything in line with this regulation makes it possible for IRMA to comply with eIDAS at a later moment easily. Since duplication of means is not allowed, we must minimize the risk that credentials are usable at multiple places at the same time. The requirements on renewal or replacement are only relevant in solutions where credentials need to be re-issued as part of the recovery process. In solutions where the old credentials will be re-used on a new device, suspension and reactivation features might be needed. Now it is only possible to fully revoke all IRMA credentials and reactivation is done by issuing new credentials again. When suspension is needed, the process must also be timely and effective and it must have the same authorization steps as IRMA account deletion currently has. Reactivation must have proper checks to make sure only the actual user can reactivate credentials. The main concepts of these flows are already available in IRMA, like for example the PIN code check. When changes are needed in these flows, the protocols must remain secure (as described in section 2.3.1 of the implementing regulation).

3.2.2. General Data Protection Regulation & ePrivacy Regulation

Besides specific requirements for electronic identification services like in eIDAS, IRMA should also meet the requirements of the General Data Protection Regulation (GDPR) [31] and the national implementation of the ePrivacy Directive [27]. The EU is planning to replace the ePrivacy directive by a regulation soon. For this regulation a proposal is already made [28]. Implementing recovery might lead to changes to IRMA or extra features might be needed. These changes and features should also comply with the GDPR and the ePrivacy Regulation.

The difference between the GDPR and the ePrivacy directive and regulation are the rights they try to protect. The GDPR protects article 8 from the Charter of Fundamental Rights of the European Union [26] about data protection, whereas the ePrivacy directive and regulation protect article 7 from the Charter [26] about privacy. Privacy and data protection partially overlap, but there are some differences. For example online communication that does not contain personal data of a user can be privacy sensitive. The other way around is also possible, for example when a user request deletion of all of his data at some company. The company can fully guarantee the user's privacy, although from data protection perspective the user always has the right to request deletion.

GDPR The foundations of the GDPR [31] mentioned in article 5 are:

- At all times Lawfulness, fairness and transparency must be taken into account
- Data is collected for a specific, explicit and legitimate purpose
- Only the minimal amount of data is collected that is sufficient for the collection purpose (data minimalization)
- Accurate and up-to-date data is used
- Data is not stored longer than necessary
- The integrity and confidentiality of the data must be guaranteed

The chosen solution for recovery should meet these principles. The guarantees that IRMA gives about these principles should not be undermined by the chosen recovery solution. These principles are important when storage of data is needed (backups) and when new communication is added. This might lead to extra personal data being processed.

ePrivacy The ePrivacy directive [27] does not add a lot of requirements to these points. The directive requires member states to implement laws to enforce security and privacy technically if possible. A important requirement is the confidentiality of electronic communication. This is important to prevent a data breach, but also communication that is not directly linked with personal data should be protected. For all communication, parties should have a legitimate purpose to process and store it and consent of the user might be needed.

The directive does not specifically focus on electronic identities, so there are no other major obligations for identity platforms to comply with this directive besides the foundations we describe above.

The draft of the ePrivacy regulation [28] is on the main points very similar to the ePrivacy directive. An important addition is that it generalizes the laws to hold for all services providing electronic communication. Besides communication the regulation also deals with user side storage of data in article 8 of the draft regulation. This article is mainly known from the browser cookie consent requirements. The IRMA app is mainly dependent on user side storage, since almost all data is stored in the app. However, when the data is needed for providing an information society service requested by the user, it is allowed. The services IRMA provides, like authentication, are information society services and a user himself decides whether to use IRMA or not, so the service is requested by the user.

Besides making sure all communication is well protected, the ePrivacy directive and regulation therefore do not involve many extra requirements that are not already mentioned in the GDPR.

4. Design Criteria for a Recovery Solution in IRMA

We discussed in chapter 3 important design principles for self-sovereign identity platforms. In this chapter we are going to use these principles to find out what is important for the design of a recovery solution in IRMA.

Definition 1. *Recovery* is defined as regaining possession of something stolen or lost[1]. In the context of IRMA we interpret this as regaining possession of credentials that were issued earlier. This can be done by getting credentials containing equivalent information newly issued or using backup and restore functionality.

In IRMA there is no strict need to regain an exact credential that was issued earlier. It is also acceptable when credentials can be re-collected efficiently for example. Therefore we will first discuss all the options we have and what options meet our requirements. This is done in section 4.1.

After that we discuss in section 4.2 the considerations we had about what is important in the design of a recovery solution. These considerations especially concern making sure the privacy and security guarantees that IRMA provides are maintained. In section 4.4 we summarized the most important considerations into design criteria.

4.1. Options for a Solution

In the introduction (chapter 1) we explained what issues users experience and why a recovery solution is necessary for the usability of IRMA. At first we step back and ask ourselves what elements are essential for a well functioning recovery process.

The IRMA app does not contain a single unique identifier of the user like the traditional identity platforms. In IRMA there is a user secret key (as explained in section 2.1). This key is not visible for the user. It is used to link credentials of separate issuers to each other and to prove commitment of the user in the process. Since the secret key is not known by others, only the user can use credentials in which his key is included. The attributes in the credentials contain the personal information (over 18, email address, etc.) and the credentials are stored on the user's device. This means that it is possible to get a clean version of the IRMA app, with a new IRMA account and a fresh secret key, and get all your attributes in there again by getting credentials newly issued.

When switching to a new phone there are three scenarios:

[1]https://en.oxforddictionaries.com/definition/recovery

1. Previous device still working
2. Previous device broken
3. Previous device lost/stolen

Ideally, a solution to switch to another device should work in all three scenarios. There are roughly three different methods to make it easier for users to switch to a new device.

Automated re-issuance of credentials into the IRMA app on a new device

Definition 2. *Re-issuance* we define as the process of an issuer issuing credentials that the particular user already retrieved earlier. There is no link between the previously issued credential and the newly issued credential. For example, when a credential expires and the user wants to retrieve a new instance of that credential, the credential must be re-issued. In the current IRMA architecture re-issuance is no alternative flow; it is exactly the same process as retrieving a credential for the first time. Currently, a user can get every credential issued as many times he wants at as many devices he likes. For every issuance, the complete authentication process of an issuer must be carried out. Each issuer determines on its own how the authentication process looks like. Since the credentials are completely new, the attributes, the secret key and the metadata do not necessarily have to be equal to a earlier issued version of that user's credential.

Automated re-issuance means that there is a process which makes it possible to retrieve all credentials a user had using the normal issuance flows of every issuer. This means nothing is fundamentally changed compared with the current approach. Credentials and attributes are still fixed to the device itself and a new phone should always get completely new credentials from the issuer. The only thing that is added is that somewhere a list is stored containing what credentials the user possessed. Using a user-side script the re-issuance of all credentials on that list is started. When the user script encounters an issuer that requires some user authentication, the user is asked to authenticate to that issuer. This has to be done for every issuer since issuers might not trust the authentication methods of other parties. This means you cannot say that only authenticating to one party is enough to re-issue all credentials. Therefore, automated re-issuance will still be rather time-consuming, for sure when more complex authentication methods are needed, like physically showing up at some location. Automated re-issuance works in all three scenarios described above, provided that the list of credentials is stored outside the device itself.

Refreshment

When the user is still in possession of his old phone, he can potentially use the IRMA credentials on the old phone to authenticate for the issuance of equivalent credentials on the new phone. The concept is similar to re-issuance as we described above with the difference that a user does not have to fully authenticate at every issuer again.

Definition 3. *Refreshment* is the process of getting issued new credentials based on authentication with the old credential. In the most basic form of refreshment, the metadata (expiry date, etc.) and the other attributes in the credential (dependent on what credential it concerns, e.g. over 18 or email address) will remain the same. Only the user's secret key might change. In this way the credentials can be used on a new device that contains another secret key.

It is important the credential's metadata and the attributes containing information do not change, because credentials might not be directly identifiable. Changing information or extending the expiry date might make the credentials inconsistent with reality. For example, a credential can be used to grant access to certain services for which the expiry date determines whether a user still has the right to access it.

When there is an attribute in the credential that uniquely identifies the user, the issuer can even consider to fully re-issue the credential by checking his database whether the attributes in the credentials are still correct. If information is not correct anymore, it can immediately be corrected and also the expiry date can be extended then.

A risk of this method is that an attacker might be able to use refreshment to re-key the secret key of a credential. This can potentially be misused to transfer a credential to another user. For example, an 18-year-old can give its credential containing an 'over18' attribute to the refreshment service and let it be re-keyed to the key of a 12-year-old user. Now the 12-year-old user can disclose the 'over18' attribute as if it is his own attribute. The design of a refreshment procedure should guarantee that a single refreshment at some issuer is part of a full refreshment process moving all credentials to a newly generated IRMA account. The credentials on the old phone should be invalidated or, when an IRMA account is used on multiple devices at the same time, the devices should be continuously synced to prevent that credentials spread out and loose the context in which they are issued.

A disadvantage of refreshment is that it does not work in the scenario of a lost or stolen phone (scenario 3), because you do not possess the old credentials anymore. Therefore it does not even fulfill our definition of recovery (see definition 1). Another disadvantage is that refreshment depends on the cooperation of all issuers. When certain issuers are not willing to cooperate, those credentials cannot be re-issued. Also, when the refreshment service of a certain issuer is not available at some moment, a user is not able to perform refreshment if it wants to regain that issuer's credential.

Backup and restore

To also make it possible to recover credentials when a device is lost or broken without introducing a complicated and time-consuming authentication processes, we need backup and restore functionality in IRMA. Automated re-issuance and refreshment have the problem that there is no proof anymore a user possessed credentials in the past when the device containing those credentials was stolen or lost. To solve this data from the old instance of the app should be backupped and stored somewhere externally from the device. This does not have to be the credential as is in the app; it must at least be a proof that the user possessed a certain credential earlier. This can be a signed message containing all credentials for example. External storage is the only way to be able to recover all data on a new device without the need of having the old device. What should be stored, depends on the recovery solution we choose.

You can do a *shallow restore* in which, simply said, the data of the old app is copied to the new device. The credentials and the user secret key on the new phone are identical to the ones on the old phone in that situation. For this you need all the IRMA app's data that was stored on the previous device in his original form. On the other hand you can do *exhaustive restore* where the user's key material and/or the credentials itself are renewed. In this method other data representations for the backup can be acceptable too.

In short The method that has our preference for the solution we are designing is backup and restore, because it is the only method that works in all scenarios for switching devices. In the following sections in this chapter we will discuss what considerations are important for a recovery solution to make sure all guarantees of IRMA can be maintained.

4.2. Design Considerations for Backup and Restore Solutions

In section 4.1 we explained why a backup and restore solution is needed to realize a proper recovery solution. In this section we discuss what elements should be considered in a backup and restore solution to keep IRMA in line with the design principles for self-sovereign identity platforms we specified in chapter 3. We discuss the design considerations per focus area. For convenience we used the terminology from the self-sovereign identity principles from section 3.1 as basis extended with usability and privacy as extra focus areas. We added privacy to deal with the GDPR requirements. We added usability since this is important for control, but from such another perspective that we separated them into two focus areas.

4.2.1. Control and Existence

Control Having recovery functionality creates the risk of an attacker installing a backup of someone else on the attacker's own device. This means identity theft might become easier. There already is a protection against losing control in the current IRMA implementation, because a Keyshare Server (KSS) is used to check the PIN code of the user. The KSS has measures built-in to prevent brute-force attacks on the PIN. Having only the brute-force protection is in some cases not sufficient. A user might use an easy PIN code or the attacker might be able to learn the PIN by observation. When the PIN code is known by the attacker at some point, there is no way for a user to prevent that its attributes are misused other than fully retracting his IRMA account. When an account is fully retracted, restoring from a backup for sure is not possible anymore. It is important to make sure measures are taken to prevent identity theft in the solution we design without fully losing the ability to restore a backup.

Existence Another possible risk is that a user might be able to transfer a credential to another user. A possible attack would be that an attacker gets IRMA credentials from another user and then overrides the credentials that are uniquely identifying with credentials that actually belong to the attacker. Due to this attack, the situation can occur that a user can prove two credentials to be from the same user, while they are not. This would make it possible to lend out your identity to others. This consequences of this attack are comparable to the re-key attack we described at refreshment in section 4.1. The only difference is that in refreshment credentials can be moved to an existing account of another user. In the attack we describe here the attacker re-uses the original account. Therefore he is required to collect new credentials for this account to be able to combine his own credentials with the credentials he took over.

Depending on the type of backup and restore solution the possible attacks for transferring credentials to other users can differ. This is determined by whether a solution has copy-protection for credentials or not; in other words whether the solution uses a variant of an *exhaustive* or a *shallow restore* respectively (see section 4.1). In solutions using a shallow restore, a user can have instances of his IRMA app on multiple devices using the same IRMA account. When having such a solution, it becomes possible to copy and distribute a credential, originating from one single issuing session, over multiple devices. In solutions with an exhaustive restore this is not possible. When a user wants to use multiple devices, he has to get issued the credentials multiple times. Then the credentials belong to different IRMA accounts and therefore they can be managed separately.

When having a shallow restore, credentials can just be copied from another user. When this is done, there is still the protection that both users now share the same PIN since they are using the same IRMA account. A limitation is that probably most users would not even have major objection into sharing their PIN with someone else, especially not when it is someone they trust or when there is a play of forces.

The limitation of sharing a PIN can also be easily circumvented by letting the actual user retrieve all his credentials for a second time. This kind of attack works in any kind of restore solution, both exhaustive and shallow, since new credentials are collected. No copying is needed at all. The new user can even change the PIN after having loaded the backup of the original user, so sharing PIN codes is also not necessary anymore. This attack can only be fundamentally protected by letting issuers log to whom they issue what credentials. This can partially harm some privacy guarantees. An issuer can see whether the same user comes back for the issuance of new credentials. Especially for non-identifying credentials this is a problem. An issuer does not have to know exactly who the user is; an issuer-specific pseudonym suffices.

Relation control and existence We combined control and existence into one section, because unless the attack scenario is different the type of measures to prevent these attacks are related. In the attack scenario concerning control the user himself is the victim and in the existence attack scenarios the relying party that is validating credentials is deceived. In both scenarios the problem is that the link between a user and an IRMA account gets lost. The guarantees about this link should remain equal when introducing backup and restore functionality. The user should be in control, but it must be prevented that a user voluntarily gives away control to realize the attacks on existence. This should be technically enforced or the consequences for a user doing this must be that high that they are not willing to do this.

The eIDAS regulation (section 3.2.1) requires identity platforms to protect against tampering and duplication. This element is included to guarantee existence and to prevent the second attack of deceiving a relying party. It also mentions that reactivation can only be done if the user or an authorized institution agrees. This also comes back in control. This means preventing the two attacks we described above are also important to comply with eIDAS.

Control is an important property of IRMA. In the current architecture only the user possesses IRMA credentials and only by the user's explicit consent attributes are disclosed, messages are signed or new credentials are issued. Because control is so important, control should also play a important role in the backup and restore features we are going to design. Extra attention should be paid that the current concept of control is not weakened by recovery.

To make sure control and existence can be guaranteed *device revocation* as a feature should be part of the restore process. Now an IRMA account can only be blocked as a whole. When a device is lost or stolen, blocking is needed in order to prevent misusage of credentials. When the only solution would be to block the full IRMA account, this would mean backup and restore would never work in the scenario a device is stolen or lost. The account should always be blocked then. Therefore we need the possibility to revoke the IRMA app on certain devices from being used. This cannot be solved by just introducing some device fingerprinting to check whether the old, compromised device is used. Simple fingerprinting techniques are easy to circumvent for an attacker and fine-grained fingerprinting techniques can give false positives. Then the actual user cannot use its device anymore too in some situations.

4.2.2. Access

Another point of attention is protection against Denial-of-Service (DoS) attacks. In this way the access to IRMA for users can be limited. Backup and restore functionality may introduce extra possibilities for this kind of attacks. As concluded in section 4.2.1, device revocation is necessary for a backup and restore solution. It should not be possible for anyone other than the actual user to revoke a legitimate device. If this would be possible, an attacker can make a user's credentials unusable on that particular device. Doing this an attacker can for example frustrate the signing process of some document. The legitimate user is not able anymore to sign a document until the blockade is lifted. This kind of Denial-of-Service attacks should be prevented.

When new services need to be created to make recovery possible, for example to store backups or to guarantee the correctness of certain procedures, there is also a new risk of limited access to these services when they are not available. In most cases this will only limit the recovery functionality. This will not harm the availability of users that still have a working app. However, in some solutions it might be needed to add checks in the regular flow to facilitate recovery. Then the availability of regular disclosing, issuing and signing sessions can be harmed.

4.2.3. Portability

It is important that users are not dependent on specific parties, for example cloud providers, while accessing their backups. When third parties can fully control the availability of the backups, the user can be forced to cooperate with those parties.

Not only parties dealing with storage of backups can be involved. In the restore process also parties might participate to ensure the procedures are executed properly. The system should also be designed to be flexible in supporting the ability to change to other parties when problems occur.

4.2.4. Transparency

Backup and restore solutions should be understandable and therefore the details of how procedures work should be open. This holds both for the procedures humans have to follow and for the source code of the IRMA app and other applications and services that are involved. Furthermore, we should aspire that the parties involved in recovery have no other intentions than just helping with making, storing and restoring backups. Proper measures should be taken to prevent that the other intentions are possible.

4.2.5. Interoperability

The recovery features should be usable for any user and it should work for all types of credentials and attributes.

4.2.6. Usability

The recovery procedures should be understandable and the steps should not be too complex for the user. Everyone should be able to do them without a steep learning curve. It must also be clear for users what their role is to maintain the security of the procedure. For example users must know what information is secret material they should protect and what they should pay attention to in order to check whether everything is still secure. Usability is important to realize control. As we described at control in section 3.1 this is important to make sure people do not perform actions by accident.

For usability it is also important that making and storing a backup can be done conveniently. A user's set of credentials only changes when newly issued credentials are added to IRMA or when credentials are removed by the user. Issuance and removal typically do not happen very often. Most users will retrieve their credentials when they start using IRMA and then they can be used for a while without a need to update them. A change in the credential set will generally not occur on a daily basis. In terms of efficiency, it is therefore not a problem when a new backup of all data stored in the IRMA app has to be made when a change occurs. The only exception are the local log files (see section 2.3). They can change every day depending on how much a user uses IRMA. This means if backups can only be made in long intervals, another solution should be found to store the log files. This is possible, since they only have to be protected against a data breach. No credentials can be restored when only having the logs. On the other side, when there is a possibility to enable making backups automatically, the logs can just be included in the regular backups.

4.2.7. Privacy

There are major privacy concerns when a backup with credentials leaks. All personal information stored in attributes of the credentials are stored in plain. The source code of `irmago` [24] shows in the file `attributes.go` that the string with information in an attribute is converted to a byte array (in the newest version shifted one bit to the left) and then stored as a big integer. This means that if you find an unprotected backup you can simply derive all personal information from the big integers that together form all credentials. In this way it is not possible to do identity fraud via IRMA, but you might have enough information to do it outside IRMA. After all, the backup probably contains all attributes a user has, so likely also very personal information is included that an attacker can use for other purposes.

This issue is particular important to comply with the GDPR and the ePrivacy regulation as described in section 3.2.2. The amount of data that is stored at other parties must be minimized. This is a hard requirement since the credentials containing all data must be included somehow. Therefore it is important to do this in such a way that as much data as possible remains confidential. According to the eIDAS regulation (section 3.2.1) this data should be stored and transported securely.

From privacy perspective in a backup and restore solution, especially the backups are an interesting element. Without any protection, backups can only be stored locally by the user itself. Otherwise there is the risk that other people can access the data, which might lead to a data breach. A problem is that letting users store data locally is very hard to enforce.

People might want to upload the backup to the cloud or store it somewhere convenient, like on their laptop. This creates the risk of other people being able to access the backups. User accounts at cloud services and personal devices of users are not always properly secured. People might use weak passwords and devices can be compromised by malware. In the case of cloud providers you also rely on these parties to properly protect the data.

Dedicated storage devices for backups that can be properly secured, require that the process of making and storing a backup becomes complicated and not very user-friendly. There is no way to make automated backups anymore, because the user always has to connect the dedicated device. Then we do not even mention the need for users to obtain such a device in the first place.

Trusted parties managing the backups is legally difficult. Those parties process a lot of extremely personal data then, requiring a high level of data protection guarantees.

Therefore it is needed to encrypt the backups in order to make sure the data is always protected properly. When this is done, it does not really matter anymore where the user stores it. He can store the data wherever he thinks is safe and convenient. This can be locally on a flash drive, somewhere in the cloud, in the user's mailbox, etc. This can even be at multiple locations and when a user has doubts he can always move the backups. This is also in line with the portability considerations we described in section 4.2.3. An encrypted backup needs an encryption and decryption key, so in a solution attention should be paid to how to deal with this and how to store the key somewhere safe then.

4.3. Using an IRMA Account on Multiple Devices Simultaneously

When backups can be exported from one IRMA app and loaded into another IRMA app, it could be a design choice to support usage of an IRMA account on multiple devices simultaneously. For example, there are users having both a private and a corporate phone or having a phone and a tablet. Users might want to use IRMA on all their devices without the need to recollect the credentials on the second device.

A drawback is that making it possible for users to use an IRMA account on multiple devices simultaneously can make the link between a user and its IRMA credentials weaker, dependent on the chosen approach. It is therefore wise to enforce that credentials can only be restored as a package and when an attribute is deleted on one device, it is also deleted on the others. In other words, the IRMA app on the different devices of a user should be kept in sync. This means that to realize this feature, extra functionality should be realized on top of backup and restore.

We described in section 3.2.1 that according to eIDAS it is not allowed to duplicate credentials. This complicates IRMA account syncing between devices even more. This does not imply it is not allowed according to eIDAS. In the current set-up IRMA credentials can only be used having the user's PIN code. This already can be seen a measure to prevent duplication. This measure is not specially indented for this and therefore rather weak, since a PIN can be easily shared with others. However, this does not mean that proper measures do not exist to support multiple devices.

We can conclude that making it possible to sync an IRMA account between different devices is not a feature that can be trivially achieved when realising backup and restore functionality. It requires extra research both legally and technically. Therefore we consider this feature beyond the scope of this research.

In section 4.2.1 we described the need of device revocation functionality. Needing device revocation combined with not realizing the supporting users to have multiple devices per IRMA account, means that device revocation should be mandatory. When a user decides to restore a backup on a new device, the IRMA app on the old device should automatically be invalidated using device revocation. Otherwise, the IRMA app would still be active on multiple devices. This automatically implies that an exhaustive restore should be used for the reasons as explained in section 4.2.1. This is an important attention point for the design of device revocation.

4.4. Design Criteria Summarized

When finding a solution for recovering IRMA attributes we have to be able to guarantee the integrity and the security of IRMA as a system. The solution should also be usable in practice. This means the solution should comply with requirements in multiple dimensions (identity guarantees, usability requirements, etc.). In chapter 3 we listed multiple sets of requirements that can be used in one of those dimensions. In chapter 4 we described the considerations related to these requirements for the specific case of recovery. In this section we summarized these considerations in seven design criteria for recovery. In every design criterion we refer to the relevant sections with detailed information.

1. Full recovery of attributes using a backup and restore strategy
   As mentioned in section 4.2.5 it is important that all types of credentials and attributes can be recovered and in section 4.1 we explain why a backup and restore strategy is necessary for this.

2. Understandability and usability
   This is described in section 4.2.6.

3. Transparency
   This is described in section 4.2.4.

4. User control
   This is described in section 4.2.1. Besides unauthorized use of IRMA credentials, it is also important for control that legitimate users should be able to get access. This is described in section 4.2.2.

5. Existence of the link between an IRMA user and the actual person
   This covers the attack vector of a relying party being deceived by users exchanging backups to fraud described in section 4.2.1. For this criterion also the points about mandatory device revocation and not allowing users using the IRMA app on multiple devices simultaneously we described in section 4.3 are important. The consequence of this is that an exhaustive restore should be an element in the solution.

6. Privacy and data protection
   This is described in section 4.2.7.

7. Portability
   This is described in section 4.2.3.

5. Recovery Potential in Other Identity Platforms

In this chapter we will give an overview of how other major identity management systems implemented recovery functionality. Koens & Meijer [18] have created a list of identity management systems. We checked the systems on this list. Not all systems have similar issues with recovery, so we only looked into systems that fulfill the criteria for a self-sovereign identity platform [1]. In this chapter we only discuss the solutions that have some kind of recovery functionality in place. Some of the solutions we discuss more extensively, dependent on the amount of technical details that was available of the particular solution.

5.1. Sovrin

Sovrin [34] is an identity platform based on ledgers. For each relation the user has with some other user (called clients), there is a key pair. The user that makes the claims has both the signing and the public keys. In this way everyone can prove using the public key that a claim is really made by that party. A claim is comparable to an attribute in IRMA. The master key of the user is always involved to make sure the claim is really linked to the particular user. The identifiers, claims, keys, etc. are stored on a public distributed ledger to guarantee the integrity of an identity wallet. From privacy and data protection perspective it is problematic to store this data in a publicly available ledger, because then all personal data of every user is publicly available for anyone. This would be a major privacy concern. Therefore users also have a private ledger to store personal data. The hashed state of the private ledger is then stored periodically on the public ledger to guarantee the integrity of the private ledgers.

Recovery In Sovrin special entities named agents are introduced [25]. These agents have the task to be an intermediate party between the user's device and the Sovrin network. Sovrin is a distributed network and therefore not all endpoints might be online at every moment. Mobile phones, for example, are not always connected to the internet. Therefore the agent handles request as an intermediate party. Every client can have multiple agents for different parts of their identity.

These agents can also be used for account recovery. For availability reasons each agent holds a copy of the relevant parts of the public ledgers of the user. The agents can also store a user's encrypted containers containing the private data. In this way if a user combines all data that are stored with their agents, then he or she can fully recover its account.

A requirement for this to work is that the user still needs its master key to be able to decrypt the encrypted containers that are stored. The master key recovery process in Sovrin is based on a web of trust[1]. The user can decide himself how many trustees he wants to have. If a user's key is compromised, it should convince a certain minimum number of those trustees to participate in key recovery. This minimum number, the threshold, can be chosen by the user itself while selecting the trustees. To perform key recovery sufficient trustees should sign a identity record transaction within a timelock period to make sure enough trustees are actively involved in the recovery process to reach the threshold.

Key recovery is only needed if a user has lost control over all of his/her devices. Sovrin supports that clients have multiple devices. If a user has multiple devices, the probability of losing the master key fully is smaller. As long as one device is still under the control of the user, it still has the master key and the master key recovery process is not needed. There are also key revocation and key rotation mechanisms to make it possible for the user to switch keys if the keys are compromised, for example when one of the other devices is lost.

5.2. CryptID

In CryptID [7] a user's identity data is stored encrypted on the blockchain using the Factom Blockchain solution. There is no way to alter your data; you should then just upload a new identity to the blockchain.

Recovery All information is stored on the blockchain, so if you lose your device you can just download the information from there. The only limitation is that the information is encrypted, so if you lose your private key, then all your information is lost.

5.3. uPort

uPort[20] is an identity platform based on Ethereum. The blockchain in Ethereum is used as for decentralized public key management and for publishing smart contracts. Smart contracts are contracts with program code in it such that the specified rules in the contracts can be enforced automatically. These smart contracts are used to capture the agreements users make into the uPort system. For example, the public key of a user is included in a smart contract and agreements over possible key revocation and key recovery can be recorded there too. In such a way it is only possible to update the public key if certain conditions, chosen by the user itself when setting up the smart contract, are met. Attributes can be issued by referring to the uPort identifier of the user where the attributes are claiming things of. The attribute is signed with the private key of the issuer.

Attributes are not stored on Ethereum. They are stored and managed by the user himself. When a relying party is requesting an attribute, the user can prove it has the attribute using a zero-knowledge proof.

[1]A web of trust is a decentralized model where users check the integrity of other users they know. For users you do not know you can trust on the judgement of someone you do know, or a person someone you know knows, etc. This is called a chain of trust; connecting two people via intermediate trustees. The ultimate goal is that every user has a chain of trust with every other user.

Recovery In uPorts philosophy, the public key cryptography should not be visible to its users, because they do not want that users have to understand the principles of public keys. Therefore they chose to have a key recovery mechanism based on recovery delegates. These delegates can be other users like friends and family, but also institutions like banks, etc. can be chosen. When a minimum number of those delegates agree with the recovery, the "controller contract" can be updated. The controller contract is a smart contract on Ethereum in which among other things the public key is published. When sufficiently many delegates agree, the user can generate a new key pair and update the key in the controller contract. This means it is not really key recovery, but key replacement.

The recovery procedure is meant to replace a user's key. It does not involve regaining the attributes the user stored privately. There is no recovery procedure described in uPort to backup those. It might be possible that the users can perform a backup for themselves by storing the attributes somewhere, but this was not described in the uPort specification at the time of writing.

5.4. SelfKey

SelfKey[16] is in approach very similar to uPort. However, SelfKey does not use the blockchain. Every user has a public/private key pair that he uses to authenticate itself to reliant parties and to sign messages. The public keys are not stored centrally or in a ledger. The public key is included in SelfKey credentials in SelfKey, called claims, and are signed by the identity owner. In this way a relying party can verify that a claim belongs to a specific user.

Recovery Storage drivers are going to be used to store the user's data encrypted externally. This is a bit similar to the agents in Sovrin. For key recovery the mechanism of uPort is used with delegates.

6. Recovery Strategy for IRMA Credentials

In chapter 4 we concluded that we are looking for a recovery solution using a backup and restore approach that fulfills the criteria we specified in section 4.4. This means we have to design a solution in which a backup can be generated from the data stored in the IRMA app on the old device and after that the user should be able to load this backup in the IRMA app on a new device. We also mentioned that device revocation should be mandatory. This means that in the process of restoring, the IRMA app on the old device should be invalidated. The concept recovery process for IRMA is visualized in figure 6.1.

The processes of constructing a backup and restoring a backup are performed largely independent from each other and can therefore be solved separately. The process of restoring consists of two elements: loading a previously generated backup and revoking the previous devices of the user. The process of loading a previously generated backup essentially is the inverse operation of constructing the backup. This means the outline of this process determined by on the decision of how a backup is generated. Therefore we include this element into the considerations for methods to construct a backup. For restoring only device revocation has to be considered separately. The solution for backing up credentials is discussed in section 6.1 and device revocation is discussed in section 6.2. Recovery solutions of other identity platforms, described in chapter 5, are partly used as a source of inspiration.

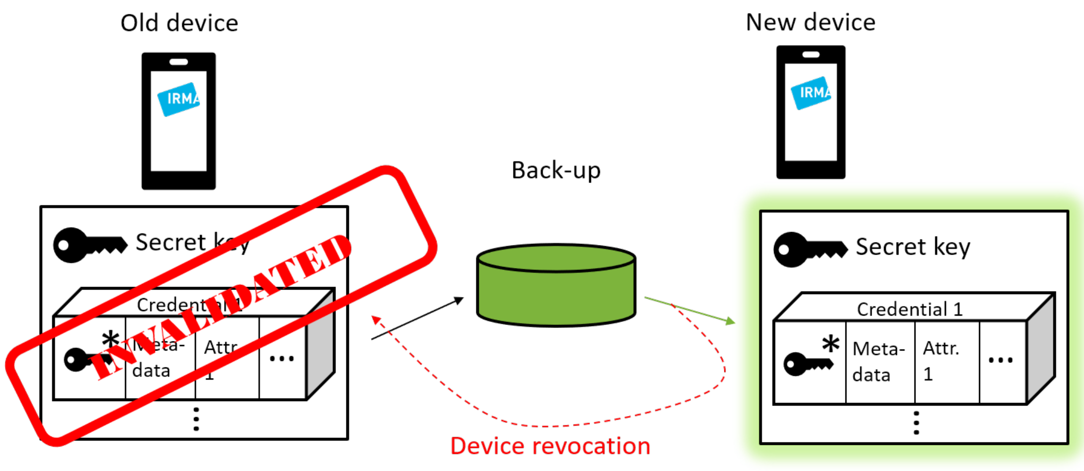


Figure 6.1.: General recovery approach

6.1. Credential Backup Strategy

An essential element in a backup and restore solution is the backup. This backup file should contain all relevant data that is stored in the IRMA app (like the secret key, all credentials, log files, etc.). A detailed overview of what data in the IRMA app is important can be found in section 2.3. We abstract from the exact location and design the solution in such a way that a backup can be safely stored anywhere the user wants. This is a design decision we make based on the portability and privacy and data protection considerations from chapter 4. Therefore, we only consider how a backup should be generated and we do not focus on where a backup can be stored conveniently.

To meet with the criteria we specified in chapter 4 we need to encrypt backups. Otherwise it is not possible to store the backup anywhere a user wants. To get encryption of a backup and to keep the user in control of its own data, an encryption key needs to be generated. The user has to manage this key himself and with that he has control over what happens with the backup. The only problem with this approach is usability. Key management is for most people not very straight forward. Encryption keys are generally too long to remember. Keys derived from a password have the limitation that the password is not used that much, so users will probably forget this. Therefore the best is to digitally or physically store the key somewhere. In this context, looking at the security usability requirements needed to realize control (section 3.1), the user must understand what he has to do to keep the key secret. Such a solution is not trivial. For example, letting the user just store a key somewhere in a file on a device is dangerous. According to Davis [8] private keys stored on a device mostly are not properly protected or the protection can be circumvented. Regular users cannot be expected to exactly know how to protect their devices properly against malware, hackers, etc.

From usability studies on PGP [33] we know that security mostly is not the main goal of users. Therefore people are not willing to invest time dealing with it. For this reason security procedures and keys are often hidden or abstracted to the user as much as possible. Eskandari *et al.* [10] looked into the usability of several key management methods for Bitcoin. They mention that when users are confronted with key management at some point, they may not understand the importance of it because they are not familiar with the concept. They only know the abstractions that are used in the application. These abstractions were often introduced with good intentions to make the app more understandable. Although, a possible consequence of this is that the awareness of the user that important key material is involved decreases. In IRMA all keys are currently hidden to the user, so our hypothesis is that the importance of protecting the encryption key of the backup will also be difficult to explain to IRMA users. Therefore the chosen solution should ideally match with the basic concepts of users to make sure they intuitively know how to protect the key.

In subsection 6.1.1 until subsection 6.1.5 we describe the different options we found that can be intuitively clear to users. We also mention the advantages and disadvantages of these solutions. In subsection 6.1.6 we judge each solution with respect to the criteria we describe in chapter 4. Based on this we conclude which option we think is the best.

6.1.1. Key printed on paper as a QR

The most user-friendly method in our opinion is storing the encryption key in form of a QR code. A regular QR code can theoretically contain at maximum 2953 bytes [13]. This is large enough to store a key. A RSA key mostly has a size of 2048 bits (256 bytes) or 4096 bits (512 bytes) and other cryptographic systems generally have shorter keys than RSA. This means there is even space left to store some extra metadata along with the key if this turns out to be necessary for the decryption process.

The QR code can be printed on paper. In this way the user has a physical document it can protect. People are used to this principle. They also do this with passports, cash money, etc. When recovery has to be done, a user can simply scan the QR again to load the key. Family can even send a picture of the paper to the user when he is not at home and needs to restore a backup. This makes it very user-friendly.

The maximum size of 2953 bytes might not be reachable in practice since the size of such a QR will be very large. This may not fit on a A4 paper. Research should be done about what the largest scannable size is when printing a QR with a regular consumers printer. We do expect that the practical size will still be large enough to at least hold a key. Otherwise some other types of QR codes can be tried like QRs with colors.

A limitation of a key stored physically is that measures should be taken to prevent it from being useful when someone else finds it. Since the information is on paper it does not even have to be stolen. A photo or a copy is sufficient to steal the information. This means the process must have two-factor authentication to prevent this scenario being possible.

The biggest disadvantage of this method is how to get the QR on paper in the first place. When a key is generated, the private key must be converted to a QR on paper and the public key must be stored on the device. The most obvious solution is that the device on which the IRMA app is stored generates the key pair. However, for most users it will not be clear how they have to print the QR from their smartphone. Even if users get this working, the QR has to be transferred to the printer. In most cases this is unencrypted or it goes via some cloud printing service. This means the key might leak in this case and it will be very hard to explain users how to mitigate this risk.

A solution could be to make a web application that securely exchanges the key pair with the device such that printing can be done from a normal laptop or desktop PC. This makes it easier to print the QR. Users can just print from their PC. With old USB connected printers this would be a safe approach, but since the emergence of WiFi-connected printers this approach has become more insecure. Furthermore, nowadays a lot of people do not have a printer anymore at all. This is a big practical limitation.

6.1.2. Key stored on USB token

Another option would be to store the key on a token instead of on paper. These tokens are designed to protect keys, so this can be made more secure than a static QR on paper. There are even tokens that support NFC, so potentially these can also be used by a mobile device. A token can also do extra access control before the key can be accessed. In this way two-factor authentication is possible.

A disadvantage is that people have to purchase a USB token to be able to store the key. Many people will not have such a token, so this is a big limitation.

6.1.3. Key stored as mnemonic phrase

To solve the transport problem of the QR and the bootstrap problem of the token, we need to have a method that makes it possible to manually transport a key from a mobile device and then being able to store it physically on paper. This has the advantage that it does not require extra equipment and that users can store the secret securely in a way they are used to. A solution is to display the recovery key on the display of the device and let the user copy the key manually. Letting user write down some number or letter code we think is not very user-friendly and users very likely make mistakes in writing down a long code.

We looked to more user-friendly mechanisms to do this and found the mnemonic phrase technique (BIP39) [22]. This method is for example used in the Bitcoin wallet to let users store their private key. BIP39 has a list of 2048 different words that can be distinguished easily. Every 11 bits of the key are converted to a number and the word on that index in the word list is displayed. The user can now simple write this ordered list of words. The words are chosen in such a way the first 4 characters are unique. In this way it is even possible to correct small mistakes users make in spelling. This makes it harder for users to make mistakes. In any situation, the amount of data that must be stored in the phrase should be limited, since every 11 bits extra makes that the user has to write down an extra word. This reduces the usability of the system.

6.1.4. Recovery key by secret sharing between one or more trustees

Instead of letting the user store a key somewhere, it is also possible to store the key at other parties. An IRMA account cannot be used on multiple device at the same time, so there is no logical way for the user to distribute his key over multiple of his own devices. Therefore, other users or third parties are needed to realize this. There are two types of trustees thinkable for this system: working with user trustees and institutional trustees.

In case of user trustees the key is stored in the IRMA app of other users. Relying on a device of just one trustee might be risky, so a possibility is that the recovery key is distributed over multiple trustees. The particular user should then choose a number of trustees. It is very unlikely that all users loose control of their device at the same time, so therefore a key can almost never get lost. To prevent that trustees can hijack someone's identity a threshold system can be used such that only when x trustees cooperate, the recovery key can be obtained.

In case of an institutional trustee the key is not stored at an individual user, but at a trusted third party whose task it is to protect the user's key. The system can support multiple institutional trustees, such that the user himself can choose which party he trusts. A disadvantage is that large institutions can be less transparent than a user trustee that the user actually knows. On the other side, institutions can deliver more support when the user has a problem, for example 24/7 support. Moreover, they can better organize proper procedures to protect keys and this can be legally enforced.

Ideally, a hybrid approach should be created where institutional trustees and user trustees are considered the same. The user himself can then decide whether to fully trust on his fellows, use one or more institutions for key management or even a combination of both.

Table 6.1.: Scorings key management solutions

	QR code on paper	Key stored on security token	Mnemonic phrase	Secret sharing	Trusted party manages key
Full recovery	+	+	++	++	+/−
Understandable and Usable	++	+/−	+	−	++
Transparency	++	+	++	+/−	−
Control	+/−	+	+/−	−	+/−
Privacy and Data Protection	−	+/−	++	+/−	+/−
Portability	++	+	++	−	−−

6.1.5. Trusted party that manages (part of) key

Instead of giving the user the responsibility of managing its key, a fixed trusted party can also be responsible for this. The user should then authenticate at such a party and if everything matches, the party participates in the recovery process. The scheme manager determines which trusted party is responsible for key management in the context of recovering the credentials that are recorded in the scheme he manages. The advantage is that the user does not have to worry where he has to store his key. His responsibility is delegated. The disadvantage is that the user also gives away the control over his backup. Those parties might misuse their trust to impersonate users.

The difference with an institutional trustee as described in section 6.1.4 is that in this solution the user has no choice which party manages his key. The trusted party is imposed to him for each credential.

6.1.6. Chosen Solution

In table 6.1 we scored each solution on how it performs on the criteria we described in section 4. We score a solution with a + if it scores positive and a − if it scores negative. When a solution scores particularly good or bad compared to the others, we indicate this with a double sign (++ or −−). The motivation for each scoring can be found in appendix A. Key management solutions do not contribute to existence, because the user can always share his key with other users. Therefore we omit this criterion in this consideration.

The options that score the best are the first three ones. These are the options in which the user is not dependent on other parties for key management. The QR code on paper performs the best on understandability and usability. However, the problem of needing a printer that you trust is problematic. This leads to both a lower score on full recovery and on privacy and data protection. The mnemonic phrase solution scores overall the best.

The only problem with the mnemonic phrase solution is that there are weaknesses with respect to control. In particular, a mnemonic phrase on paper can be stolen or copied. To perform well on all criteria, two solutions should be combined. Therefore we will introduce a two-factor authentication solution.

The first factor would be to convert a key to a mnemonic phrase. This key will then be used as the first encryption layer over the plain data of the backup. The user has this key in his own control. This means that after encryption it does not really matter anymore where the data is stored. It cannot be used without the user's key.

The second factor should be a solution that at least performs well on the particular element of control that makes stealing or copying the key become harder. There are three solutions by which this can be achieved: a key stored on a security token, secret sharing and a trusted party that manages the key. The key stored in a security token solution drops out for the same reason it dropped out as a solution for single-factor authentication: most users do not have such a token. This means that we have two options left that both have its advantages and disadvantages. The scores on control are not really relevant anymore for a second factor, because these objections are already taken care of in the first factor (i.e. the mnemomic phrase). Secret sharing scores better on realizing full recovery and gives a bit more transparency and portability, whereas a trusted party is more user-friendly.

The main disadvantage of a trusted party concerning full recovery is that it depends on the trusted party which credentials can be recovered. The idea of the trusted party approach is that per scheme manager a trusted party is appointed that deals with recovery guarantees. This means that a scheme manager can decide to not support recovery and then users cannot recover credentials that are included in these schemes. For example, this will be the case for demo credentials. It is also possible that different scheme mangers use different trusted parties, requiring the user to perform authentication at multiple trusted parties. This might influence usability.

However, currently in IRMA we only know one scheme manager, namely the Privacy by Design Foundation. This means that currently there is no risk of differences between schemes. The only disadvantage is that a user would lose demo credentials, but since these credentials are not intended to be of any value this should not be a problem.

This means in the current set-up only transparency and portability make the difference between both solutions. Secret sharing gives the user more freedom in which parties he trusts with a part of this key. This option is therefore favorable above one single trusted party. For usability you could then consider only allowing institutional secret sharing such that a user never has to contact multiple trustees in order to perform recovery.

Although, when turns out that a trusted party is needed anyway to guarantee existence, then this trusted party can act as the second factor as well. In this case, it would be cumbersome to also include secret sharing. In fact, you would then create three-factor authentication, since the trusted party's agreement is also necessary for the restoring process. This is maybe a bit too much of a good thing. In this scenario, a trusted party actually is an acceptable alternative. The existence guarantees are considered in the device revocation process (section 6.2).

6.2. Device Revocation Strategy

In chapter 4 we mentioned that part of the design of a backup and restore solution should be mandatory device revocation. This to enforce that old devices can no longer be used when a new device is introduced. To realize this we have to find a solution to implement an exhaustive restore. This means something should be done with revocation or replacement of key material on the old device of the user in order to make sure that a device can be blocked remotely. This key revocation or key replacement should be included in the restoring process. In this way a backup can only be restored when the old device is revoked first. In this way the user has more control over the devices and the security of the key is less reliant on the PIN check of the keyshare server.

In subsection 6.2.1 we first describe the several technical solutions that can be used. These solutions all have their own characteristics. A device revocation solution will be a combination of a technical solution to perform revocation combined with a approach of how this solution is managed and by whom. In subsection 6.2.2 until subsection 6.2.5 we explain the different approaches we came up with. In some approaches only a subset of technical solutions can be used to properly realize them. Therefore we also describe which solutions can be used to implement the approach. In subsection 6.2.6 we judge these options to the criteria we specified in chapter 4 and explain which solutions we think are the best. In subsection 6.2.7 we discuss which option we recommend for an implementation taking into consideration the current IRMA architecture and the credentials that currently exist.

6.2.1. Overview of Technical Solutions for Revocation

For device revocation in general there are two options: a central party should block the particular device or the credentials themselves must be revoked (credential revocation).

Credential revocation is a concept that has been subject to prior research. Several solutions were developed to realize this for Idemix credentials. Lapon *et al.* [19] did an analysis of these solutions. Based on this analysis we mention some of the different solutions that we thought that are promising for credential revocation in IRMA. We do not mention the solutions that have limited overhead, since they break some of the privacy guarantees by design. The possible options in the field of credential revocation are described in option A until C.

In option D we discuss the option of a central party being responsible for the revocation process.

A. Time-based credential updates An option to make revocation possible for all credentials is adding epochs to credentials. The user should contact the issuer in each new epoch to update his credential. It is then possible for issuers to revoke credentials by not cooperating anymore in updating a user's credentials. A revocation list should be maintained and all issuers must make sure credentials on this list are revoked. To maintain multi-show unlinkability it is important that credentials are not refreshed on usage, but on a random moment at the beginning of a new epoch. Otherwise the issuer might be able to detect usage of his credential.

The advantage of this solution is that the checks needed for this solution are already implemented in IRMA. IRMA already knows expiry dates for credentials, so this expiry date can just be shortened. This naive solution is not very efficient. Now a full new credential should be achieved when it is expired, so to make it efficient a feature should be added to update credentials non-interactively. This means that a user can just download updates of his credential at the issuer without needing a complex updating process.

A disadvantage is that periodic contact is needed between the IRMA app and the issuers. This makes it possible for attackers to learn which credentials a user uses by doing network analysis. An attacker may know the IP addresses of the issuers and it can check the IP addresses a user contacts. To partly solve this TOR or some other type of mix layer should be used to hide the exact IP addresses users contact using IRMA.

B. Verifier checks revocation list Another option is that before accepting a proof made by the user, at first it is checked whether the used credential is not on a revocation list. Nothing is changed to the validity of credentials on user side. Additionally to the proof, the user has to add some token which the verifier can compare with the revocation list. The revocation list should be made up by a recovery authority or another mechanism can be used to agree upon a revocation list between users, issuers and verifiers, like a distributed ledger.

The users cannot just disclose some pseudonym to the verifiers to let him check the revocation list, because this would break unlinkability for non-identifying attributes. Another risk of some implementations is that some create backwards linkability. This means that before revocation the privacy can be guaranteed, but once revocation is needed all previous sessions become linkable. Therefore an important prerequisite is that a solution is used that does guarantee the user's privacy, even after revocation. A disadvantage of solutions that have these guarantees is that they tend to be rather computationally inefficient, especially when the number of revoked credentials increases. This may cost the verifiers much computing power. On the bright side, the complexity for the IRMA app on the user side does barely change.

C. User proofs not being on revocation list Nowadays smartphones have a reasonable amount of computing power available, so to save verifiers from having to run complex verification algorithms the IRMA app might as well be able to take this responsibility. The user then proves that his credential either is or is not on the revocation list, depending on whether it is a whitelist or a blacklist. The verifier can simply verify this proof and does therefore not have to self check the full revocation list. This saves a lot of time.

A whitelist approach, so a list of valid credentials, in this case is more efficient. A user can then just point to his entry in the whitelist and prove that he actually represents that entry. With blacklists the user has to prove his credential is not on the list. This is a bit more complex, but still possible.

To guarantee the privacy of the user, revocation lists are not just lists with identifiers. The lists contain anonymized signatures. This means that when a new credential is added to the black- or whitelist, the issuer has to completely regenerate the list. This costs a lot of computing resources from issuers. Therefore, the most efficient solution accumulates all signatures into one value, such that new signatures can just be added to the previous value of the accumulator [21]. A user then has to prove that the value of his credential is or is not (depending respectively on white- or blacklisting) part of the accumulated value. This solution is efficient for both the issuers and for the verifiers. The computational load is on the user. Research has to be done to find out whether a smartphone app can deal with this complexity.

D. Central party checks revocation The final option is that a central party fully manages the device revocation process. In this approach in any proof or for any access to the IRMA app agreement is needed from the central party. When a user blocks his device, the central party will simply not participate anymore in any of the operations initiated by the blocked device and therefore none of the credentials on the device can be used anymore.

This option is the most straightforward solution and it is easy to realize. A disadvantage is that no specific credentials can be revoked, because then the central party can track usage of credentials. Another disadvantage is that the system becomes reliant on one party which can become a single point of failure.

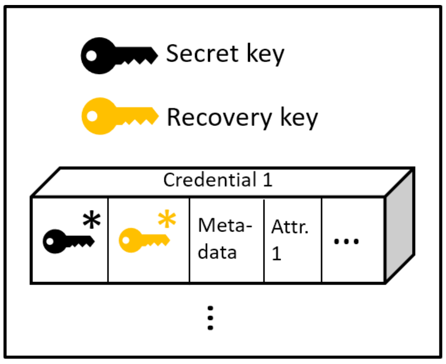
6.2.2. Extra Key in Credentials

In any solution described in section 6.2.1 the responsible party/parties should be convinced the user that is restoring a backup actually is the right user. It is not possible to do this using the existing secret key, because this one might be compromised. A possibility is to include more than one key in each credential.

Option I: multiple secret keys You could add extra secret keys to an credential to make it possible for a user to switch to another secret key when one of the other secret keys is compromised. Checks have to be added to make sure previous keys cannot be used anymore. New devices can just use one of the keys that are not revoked yet. Important is then that only one of the keys is known to a certain device and that only one key can be used at the same time.

For this solution to work the way of making the signature of credentials should be altered. Now the value of all attributes in a credential is included in the signature of the credential. When multiple secret keys are included as attributes in the credential and the old signature method would be used, then all keys are required to produce a valid proof. This means all secret keys have to be known in order to use IRMA. In this way, multiple secret keys would not have any advantage, because essentially the multiple keys form one single large key then. Therefore, instead of one signature, multiple signatures must be included in the credentials to make this option work. For each extra secret key also an additional signature is needed. When a new device is configured containing another secret key, then that device can simply use the corresponding signature to proof the credential is valid. One of the options described in section 6.2.1 should be used to revoke the secret keys that are already used.

Option II: revoke-and-refresh Another approach would be to include a user secret key and a recovery key in each credential. This approach is visualized in figure 6.2. You could then do an automated refreshment procedure. Each issuer should have a service where you can disclose your credential, prove that you have the backup key and then the issuer issues a copy of the credential you provided with a new user secret key and a new backup key. The restore procedure of backup and restore then is a refreshment procedure. In this way you prevent the problem we described in section 4.1 that refreshment on its own cannot be used when a device gets compromised. The backup key is not stored on the device, but somewhere else. Therefore that key is not compromised when a device gets lost and can therefore still be used to prove ownership of a credential. In this solution the number of



* Key in credential can be a combination of the user key and a keyshare managed by some trusted party.

Figure 6.2.: Credential outline when using a recovery key

recovery attempts is not restricted, but it is more dependent on the cooperation of issuers. All solutions from section 6.2.1 can in theory be used to revoke the old credentials that have been refreshed. However, since all issuers have to be contacted for refreshment anyway, solution A until C are more obvious to choose.

Drawbacks A drawback of option I is that only a limited number of recovery attempts is possible; exactly as many as there are extra user keys in a credential.

In case of option II, for revoke-and-refresh solutions it is hard to streamline all refresh procedures that are part of a restore. For the refreshed credentials a new IRMA account should be made with a new secret key. Issuers must know for sure the new secret key cannot be freely chosen by the user. Otherwise a malicious user can re-key credentials from multiple users to the same key, making it possible to combine them. Issuers themselves cannot supply randomness to this key, because the key should be the same across all issuers. To solve this, a trusted party should be introduced that makes sure all issuers use the same secret key in a refreshment. When a trusted party is introduced to guarantee existence, the question could be asked what authentication at every individual issuer then adds. The only thinkable scenario for which this solution (extra key option II realized with solution D) can be useful is when issuers do not fully trust the trusted party and want to perform authentication themselves too before a credential is refreshed. For revocation of the previous device they are always reliant on the trusted party in this scenario, so a lack of trust in the trusted party will remain problematic.

What makes this approach especially difficult is that every issuer should also host a refreshment service. This adds another level of complexity for parties that want to issue a credential. When issuers refuse to set this up, credentials from that issuer cannot be recovered. Also, when there is a disruption in the refreshment service of one issuer, the restore process as a whole can be frustrated.

If option II has the preference, there are two possible approaches. When no trusted party is used, the issuers must accept the risk that existence will be weakened a bit. When a recovery authority is used as a trusted party, existence can be guaranteed, but the portability of the solution gets less. For solutions with a distributed ledger (e.g. blockchain kind of solutions), the extra recovery key solution is unnecessary, since the issuer has the ability to check for malicious behaviour himself by consulting the ledger. Above we pointed out that when a trusted party solution is chosen it is important that there is no lack of trust in this party among issuers and that solution D combined with refreshment at every single issuer is only useful in scenarios when there is a lack of trust. Given this contradiction, we think only solution A until C can be considered useful for this approach. Therefore the weakened existence because of the lack of a trusted source is inherent to option II.

6.2.3. Revocation via Keyshare Server

When a keyshare server is used for a credential, not the full secret key of that credential is stored on the device on which the IRMA app is installed. Only the user part is. The other part of the key is managed by the keyshare server. The way it works now is the following: $k_{secret} = k_{user} + k_{kss}$. This is computed using a multi-party zero-knowledge proof. When a device gets compromised, only the k_{user} part is compromised. The k_{kss} and therefore also the k_{secret} are still safe. k_{secret} is the part that is included in the credential; nobody knows this key fully. A solution for device revocation would be to support re-keying of this secret key in IRMA. The keyshare party then becomes a central revocation authority as described in option D of subsection 6.2.1. When introducing a new device, a delta should been added to the user secret key. The opposite can then be done to the keyshare part of the key to make sure the combination is still the same. For some generated δ the equation would then be: $k_{secret} = (k_{user} + \delta) + (k_{kss} - \delta)$. The left part between brackets becomes the new user secret key and the right part becomes the new keyshare part without having to change the k_{secret}. The explanation of the changes in the current protocol is explained at the end of this section.

To realize this, the keyshare server should at least be extended with a re-keying feature such that the user can show he has knowledge over the PIN code and some additional authentication factor. The additional factor is needed, because we consider a PIN only too weak for an extensive operation as recovery is.

Doing device revocation at the keyshare server can be an issue for portability. The user cannot choose which keyshare server he wants to use for a certain scheme. This can only be chosen by the scheme manager. However, this problem is inherent to the existence of the keyshare server. Currently, the keyshare server has to be in place to guarantee the PIN check to be valid anyway. Given that adding device revocation with a device key does not add any dependencies to this model.

A consequence of this is nevertheless that device revocation cannot be used by scheme managers that have no keyshare server. In theory, the concept of a keyshare server is a abstract concept for splitting a IRMA secret key over multiple locations. It can even be splitted over more than two parties. This concept can for example also be used by a recovery authority that is put in place or even a system with multiple recovery agents. Therefore choosing for a architecture like this does not automatically imply a central keyshare server is required. Other architectures can be built too. This means that even in this solution portability is still an option if scheme managers want to achieve this.

Security The change does not effect the security of the protocol for external attackers. For the calculation we just add the device key k_{device} as a constant. For third parties this just works as extra key entropy. When the $k_u ser$ gets compromised, the device key can

Usual protocol for zero-knowledge proofs of hidden attributes [4] using the Fiat-Shamir heuristic [11] as described in the IRMA keyshare protocol [15]:

1. User wants to show that it knows m. Given a publicly known $P \in QR(n)$, the user chooses a generator R from $QR(n)$ such that $P = R^m$.
2. Verifier sends random η to prover
3. Prover
   a. generates random ω
   b. computes commitment $W = R^\omega$
   c. computes challenge $c = H(P, W, \eta)$ where H be a hash function
   d. computes response $s = cm + \omega$
   e. sends c and s to the verifier
4. Verifier computes $W' = R^s P^{-c}$ and $c' = H(P, W', \eta)$. He can then verify whether $c = c'$.

This proves the knowledge of m since the only difference between c and c' is W and W' in the appliance of the hash. For W and W' the following holds:

$$W' = R^s P^{-c} = R^s R^{-mc} = R^{cm+\omega} R^{-cm} = R^{cm+\omega-cm} = R^\omega = W$$

Since P is publicly known, this can only be right when the actual m is used in s and therefore the prover cannot cheat.

Keyshare protocol for zero-knowledge proofs of hidden attributes (multi-party proof) as described in the IRMA keyshare protocol [15]:

Secret m is now a combined secret from two parties, in case of IRMA the user and the keyshare server. Thus: $m = m_u + m_{kss}$ where m_u is the secret of the user and m_{kss} the secret of the keyshare server.

The keyshare server is therefore added as a party in the protocol. The protocol changes in the following way:

1. User wants to show that it knows m. Given a publicly known $P \in QR(n)$, the user chooses a generator R from $QR(n)$ such that $P = R^m$.
2. Verifier sends random η to prover
3. Prover generates random ω
4. Keyshare server generates $W_{kss} = R^{\omega_{kss}}$ and sends this to the user
5. Let H be a hash function and $W = R^{\omega}$, prover computes challenge $c = H(P, WW_{kss}, \eta)$ and sends c to the keyshare server
6. The keyshare server computes $s_{kss} = c * m_{kss} + \omega_{kss}$ and sends this to the user
7. User computes $s_u = cm_u + \omega$ and sends c and $s = s_u + s_{kss}$ to the verifier.

   In case of issuance s_u and s_{kss} are both sent to the verifier, combined with a signature of the keyshare server on s_{kss}. This is needed to guarantee that the actual keyshare server is used to calculate the secret key and not a local instance of the keyshare server.
8. Verifier computes $W' = R^s P^{-c}$ and $c' = H(P, W', \eta)$. He can then verify whether $c = c'$.

   This is right because:

$$\begin{aligned} W' &= R^s P^{-c} \\ &= R^s R^{-mc} \\ &= R^{s_u + s_{kss}} R^{-c(m_u + m_{kss})} \\ &= R^{cm_u + \omega + c*m_{kss} + \omega_{kss}} R^{-cm_u - cm_{kss}} \\ &= R^{cm_u + \omega + c*m_{kss} + \omega_{kss} - cm_u - cm_{kss}} \\ &= R^{\omega + \omega_{kss}} \\ &= R^{\omega} R^{\omega_{kss}} \\ &= WW_{kss} \end{aligned} \tag{6.1}$$

   Thus $c' = H(P, W', \eta) = H(P, WW_{kss}, \eta) = c$

Variant with additional device key

Besides m_u and m_{kss} we now add a device key k_{device} which is a shared secret between the keyshare server and the user's current device.

1. User wants to show that it knows m. Given a publicly known $P \in QR(n)$, the user chooses a generator R from $QR(n)$ such that $P = R^m$.
2. Verifier sends random η to prover
3. Prover generates random ω
4. Keyshare server generates a random ω_{kss} and computes $W_{kss} = R^{\omega_{kss}}$ and sends W_{kss} to the user
5. Let H be a hash function and $W = R^{\omega}$, prover computes challenge $c = H(P, WW_{kss}, \eta)$ and sends c to the keyshare server
6. The keyshare server computes $s_{kss} = c * (m_{kss} - k_{device}) + w_{kss}$ and sends this to the user
7. User computes $s_u = c * (m_u + k_{device}) + \omega$ and sends c and $s = s_u + s_{kss}$ to the verifier.

   In case of issuance s_u and s_{kss} are both sent to the verifier independently combined with a signature of the keyshare server on s_{kss}. This is needed to guarantee that the actual keyshare server is used to calculate the secret key and not a local instance of the keyshare server.
8. Verifier computes $W' = R^s P^{-c}$ and $c' = H(P, W', \eta)$. He can then verify whether $c = c'$.

Proof

$$\begin{aligned}
W' &= R^s P^{-c} \\
&= R^{s_u + s_{kss}} P^{-c} \\
&= R^{(c(m_u + k_{device}) + \omega) + (c(m_{kss} - k_{device}) + \omega_{kss})} P^{-c} \\
&= R^{cm_u + ck_{device} + \omega + cm_{kss} - ck_{device} + \omega_{kss}} P^{-c} \\
&= R^{cm_u + \omega + cm_{kss} + \omega_{kss}} P^{-c}
\end{aligned} \tag{6.2}$$

Now the equation is equal to the proof of the regular keyshare protocol in equation 6.1 and therefore we can conclude that $c = c'$ must still hold.

be changed. When the key space of the device key is large enough, the security can be guaranteed.

The only weakness that is added by this approach is that refreshing the device key does not work in case of a hostile keyshare server. When the keyserver server gains the knowledge of m_u, it will know all secret values from then since k_{device} is a shared secret from the user and the keyshare server. However, the concept of the keyshare server is that it is a trusted party that guarantees the security of IRMA. The scenario that the keyshare server cannot be trusted, is therefore not very realistic. When a part of the data of the keyshare server gets temporarily compromised and the server can still be trusted after that moment, the concept would work, because then a newly generated delta will not be known by the attacker. In all the cases, the user always has the option to not use device revocation and simple start a complete new IRMA instance when it does not have trust in the keyshare server at all.

6.2.4. Credential Revocation Based on a Distributed Revocation List

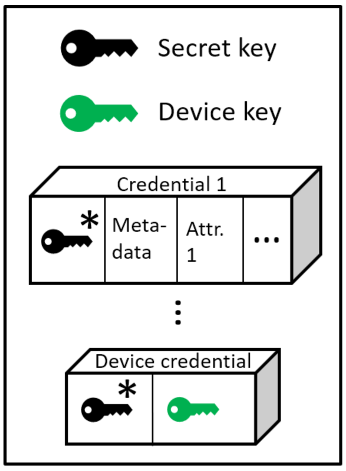
As an alternative to deal with device revocation without fully depending on a central authority, like the solution as described in subsection 6.2.3 does, we can also do the opposite and use credential revocation to revoke every credential separately. In this solution we put the responsibility to prove whether a device is not revoked either on the user or on the verifier. From technology perspective this means that option A until C from subsection 6.2.1 can be used to realize this approach. Refreshment will be needed to transfer credentials to the new device. Therefore this solution also is of the type revoke-and-refresh.

Instead of making sure the IRMA app on the old device cannot be used anymore, the credentials can be revoked. This means that the credentials in theory can still be used to produce proofs, but an additional check is added to the verification algorithm to make sure that the credentials used are not yet revoked. This can be done by either the user somehow proving that his credentials are still valid or the verifier somehow checking that the used credentials in the proof are not revoked yet. In the first scenario the verifier also has to check that the extra proof the user makes is correct.

Unless the responsibility for revocation is distributed over the issuers, also in this solution some kind of authority is needed, because when a certain device must be blocked, somehow the user should authenticate and after that the revocation list should be updated. This cannot be done by some central authority, because then we are actually creating the solution from section 6.2.3 with additional overhead. This means the revocation list should be maintained in a distributed way. For example, it could be possible to realize it via smart contracts on a distributed ledger. The key element of the distributed revocation list is that it provides a single truth between users, issuers and verifiers about which device should be revoked and what the new device is that replaces it.

Credential revocation as a solution for device revocation has the advantage that credentials can also be revoked for other reasons. Issuers will have more control over the credentials and therefore can do more checks to guarantee existence of the user.

A drawback is that revoking every credential individually creates a dependency on all issuers. Only issuers can know what credentials were issued, so they are also the only party that can revoke them again. This means issuers are going to be required to keep records of the credentials they have issued. Therefore extra data should be stored. Additionally, the revocation process becomes more complex. When a user revokes his device, he has



* Key in credential can be a combination of the user key and a keyshare managed by some trusted party.

Figure 6.3.: IRMA wallet outline when using device credentials

to somehow publish this on the distributed revocation list and then all issuers have to be notified. The issuers all have to individually take action to make sure credentials get revoked and new credentials are issued for the new device of the user. Finally, when a issuer decides not to implement it, credentials of this issuer cannot be restored.

6.2.5. Revocation Using a Device Credential

In an attempt to solve the disadvantages of both indivual credential revocation (section 6.2.4) and revocation using the keyshare server (section 6.2.3), we came up with an intermediate solution: the device credential. The procedure is that when a user initializes a device, the scheme manager's revocation authority grants a device credential to the user stating that a certain device identifier or device key is allowed to do IRMA activities on behalf of a certain secret key. The user can show this credential at every issuing, disclosing or signing event to prove that the device was set-up properly. The validators and issuers have the responsibility to check that this is true. The credential structure that is needed for this model is visualized in figure 6.3.

Using this solution, the restore process is not dependent anymore on all issuers. The revocation authority himself can decide to revoke the device credential and then verifiers can reject the proof if revoked device credentials are used. This makes the process less complex and it also makes sure all credentials within one scheme either support or do not support backup and restore. This makes it much clearer.

The methods that can be used to perform device credential revocation can be the same as for any credential using option A until C. Additionally, also option D with a central server is an option here. For this solution, in the device credential an device key attribute should be included then and this device key is stored by a server. This should be a new keyshare that exists besides the keyshare on the secret key that currently exist. The idea of a keyshare server here should not be confused with the solution we described in section 6.2.3. In that solution the keyshare server takes a keyshare of the secret key and then the device key is used as a delta on this secret key. In this solution, the device key is fully separated from the secret key. The user does not know the device key such that when the server blocks it, the key cannot be accessed and therefore the device credential cannot be used anymore.

6.2.6. Decision Considerations

In this section we compare the different combinations of recovery approaches and technical solutions with each other based on the criteria we drew up in chapter 4. Similarly to the comparison we did for key management solutions, we made a scoring for the device revocation solutions. A solution is scored with a + if it scores positive and a − if it scores negative for a certain criterion. When a solution scores particularly good or bad compared to the others, we indicate this with a double sign (++ or −−). The motivation for each scoring can be found in appendix B. A summery of the scorings is visualized in table 6.2.

At first we consider the approach in which extra keys are added to each credential (section 6.2.2).

Option I to include one or more extra secret keys, makes it only possible to recover a limited number of times. This option is too limited, because it is hard to explain that at some point a user cannot recover anymore due to some technical issue. Therefore it does not satisfyingly meet the criterion full recovery.

Option II, where a recovery key is included as an extra attribute in each credential, performs better since there is no limitation on the number of recovery attempts. The possibility to recover in this solution is dependent on the willingness of issuers to cooperate. When they do not do this or when they have disruptions in their services, the user is not able to recover all of his credentials. Nevertheless, scheme managers can enforce issuers to support refreshment and issuers also get in return extra options to revoke credentials for other reasons.

The major drawback of option II is that existence is weakened. To solve this, it is necessary a single source for information is created such that users cannot choose a secret key that already exists. The recovery key in credential (option II) approach does not have such a source. As explained in section 6.2.2 solutions with a trusted party or a distributed ledger in combination with a recovery key per credential are cumbersome mechanisms and therefore we consider those as not applicable.

The three options that are left all have their pros and cons. The only option that scores well on portability is credential revocation based on a distributed revocation list. In this approach the user should publish in the distributed database that he wants to revoke his device and then this has to be propagated to all issuers before revocation can take place. There is no single party responsible where users can complain. This means for transparency of the system this is a highly complex solution. Besides from that it is also unclear from a technology perspective whether a distributed solution can be realized that can deliver the privacy and security guarantees of IRMA and also can be streamlined nicely in the processes for a restore. Thus, this mainly is a theoretical solution for which it is is not known whether it can be realized. Therefore we do not further consider this option. Technical solution D in this case is marked as ‘not applicable’, since this solution is not a credential revocation technique.

Now there is still the choice between issuing special device credentials (section 6.2.5) and applying a delta to the keyshare of the keyshare server (section 6.2.3). The approach of

Table 6.2.: Scorings device revocation solutions

Approach:	**Recovery key in credential (option I)**				**Recovery key in credential (option II)**			
Technical solution:	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Full recovery	–			– –	+/–			
Understandable and Usable	– –	–	–	–	+/–			
Transparency	++	+	++	+/–	++	+	++	
Control	–			+	–			n.a.
Existence	+	++	++	+	– –	–	–	
Privacy and Data Protection	– –	–	–	+	– –	–	–	
Portability	+			–	+			

Approach:	**Via keyshare server**				**Credential revocation based on distributed revocation list**			
Technical solution:	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Full recovery				+	+/–			
Understandable and Usable				++	+			
Transparency				+/–	–	– –	–	
Control	n.a.	n.a.	n.a.	+	+/–	–	+/–	n.a.
Existence				+	+	++	++	
Privacy and Data Protection				+	–	+/–	+/–	
Portability				–	+/–			

Approach:	**Device credential issued by trusted party**			
Technical solution:	**A**	**B**	**C**	**D**
Full recovery	++			
Understandable and Usable	+			
Transparency	+	–	+	+/–
Control	+			
Existence	–	+/–	+/–	+
Privacy and Data Protection	+			
Portability	–			

using a device credential gives a scheme manager more freedom in what technology he wants to use for device revocation. In fact all technical solutions (A until D) can be used to realize this. The technical solutions that perform the best when considering the scores from table 6.2 are the user proving his device has not been revoked (solution C) and a trusted party approach (solution D). Solution C performs better on transparency and solution D scores better on existence. On the other criteria they score equally.

An important remark: the approach using the keyshare server scores exactly the same as the device credential approach with technical solution D. This means that when there is a high preference for technical solution D, it does not matter whether the keyshare server approach or the device credential approach is used.

6.2.7. Chosen Solution

Currently, the keyshare server is still needed to check the PIN of the user. This means when device revocation in IRMA would be realized with a device credential approach using solution B or C, the keyshare server does not become superfluous. Implementing recovery via solution B or C costs much work, since these solutions require preliminary research about how these techniques can be applied to IRMA. On top of that the current flows in IRMA become less efficient since extra proofs are needed. This means that having credential revocation only adds complexity without making IRMA less dependent on the keyshare server. Therefore we think it is acceptable to give the keyshare server also the role of trusted authority for recovery. The keyshare server only has to cooperate for making possible device revocation and does not have to learn extra data of users. This means this approach does not extend the power of the keyshare server much. We mainly use functionalities the keyshare server already has in IRMA nowadays.

In section 6.2.6 we concluded that for the quality of the solution it does not matter whether a device is revoked via the keyshare directly (the approach from section 6.2.3) or via an intermediate device credential (the approach from section 6.2.5 using technical solution D). Therefore we choose for the solution that is most efficient and can be implemented the easiest. This is the approach using the keyshare server directly as described in section 6.2.3.

A disadvantage is that schemes without a keyshare server cannot use recovery. In the current landscape of credential types this is not a major problem, since IRMA only knows two schemes: the scheme for demo credentials and the scheme from the Privacy By Design Foundation (PBDF) containing all real-life credentials. PBDF's scheme uses a keyshare server, so all these credentials are supported. Only credentials from the demo scheme cannot be recovered. These demo credentials are only used for testing or for demos and have no real value. Therefore we consider this drawback not having major consequences.

When in the future credential revocation might be implemented in IRMA in some kind of way, there is still the possibility to replace server side blocking by the keyshare server with the revocation of a device credential that states that a certain device is allowed to be active. A central party is still needed to issue and revoke device credentials. This means the device management functionality we have to design for the keyshare server can be re-used.

7. Implementation

In order to show that the chosen recovery strategy as described in chapter 6 is possible, we designed a proof-of-concept to show that the design functions in practice. In this chapter we first explain the design we made to implement recovery. This concerns some changes in the calculations of proofs, the design of protocols for the new features (making backups, restoring a backup and the initalization of key material) and the method how a backup should be generated. After that, we explain how we implemented this design in the current IRMA infrastructure. When we discuss source-code, we only refer to the name of the repository in which the particular code can be found. An overview of the names and the URLs of the source-code repositories can be found at the end of this chapter.

7.1. Proof Calculation Changes

In section 6.2.7 we described that we are going to implement mandatory device revocation via the keyshare server. This means we have to add a device key to both the IRMA app (to be specific the IRMAGo library) and the keyshare server. The way the proof calculations have to be changed is described in section 6.2.3.

IRMAGo The IRMAGo library contains the core implementation of IRMA. This library is used by the IRMA app in the background to deal with the management of credentials and performing IRMA sessions (disclosures, issuance, etc.). Therefore, to make the device key become a part of IRMA proofs, we had to make some changes to this library. In the first place we added support for storing the device key. The source code of this is in the file `irmaclient/keyshare.go`. Secondly, the device key must be included in the calculation of the secret key. Therefore in the file `irmaclient/client.go` we added the function `findCredentialSecretKey` to combine the device key and the already existing secret key. This function requires as parameter an identifier of the scheme manager of which credentials are going to be used in the proof. This is needed in case the app knows multiple keyshare servers. The user might have multiple device keys then; one for each keyshare server.

The secret key is stored separately and usage of the secret key was therefore easy to localize in IRMAGo. This means finding where the device key should be added to the secret key was rather easy. The essential programming code for this can be found in figure 7.1. We needed to insert this code in the function `credential` that composes a credential. This function can be found in the file `irmaclient/client.go`.

We implemented it in such a way that the keyshare server modifies the keyshare part of the secret key directly when calculating its commitment. Therefore, no further change is needed in the existing IRMAGo code to apply the device key other than the change we described above. The code that deals with storing the device key can be found in the `keyshareServer` struct of the `irmaclient/keyshare.go` file. By default we set the device

Figure 7.1.: IRMAGo code to calculate modified secret key

```
// Adapts secret key with the right delta when the scheme managers
    keyshare has been changed
func (client *Client) findCredentialSecretKey(
  smid irma.SchemeManagerIdentifier) (key *big.Int) {
        //Scheme managers without keyshare server use the direct
           secretKey
        if !client.Configuration.SchemeManagers[smid].Distributed
           () {
                return client.secretkey.Key
        }

        kss := client.keyshareServers[smid]

        return new(big.Int).Add(client.secretkey.Key, kss.
           DeviceKey.Key)
}
```

key to 0. This means existing IRMA apps do not have to negotiate a device key first. Just the zero constant can be given to this field initially. This is not an issue for security, because at initialization the user part of the secret key remains within the storage of the IRMA app. In section 2.3 we explained that this storage is not accessible by anything else than the IRMA app on non-rooted devices. The secret key is also included in the backup files that we are going to generate. The backups however are fully encrypted and can only be decrypted with help of the keyshare server. The keyshare server functions as second factor in the authentication. This means the keyshare server is able to notice when a backup is going to be decrypted and as a prerequisite it can demand the user to first negotiate a new device key. This means that the zero initialized device key can only be used until a backup has been decrypted. From this moment on the secret key might be known outside the IRMA app too, so only from then the fresh key material from the device key is necessary.

Keyshare Server Java Implementation As mentioned in the paragraph above, the change to combine the device key and the secret key in each credential is not a very complex change. This also holds for the changes to the keyshare server. In the `User.java` file a getter was already implemented that gets the keyshare of the particular user from the database. This function is used at all places where the keyshare is accessed. This means we could simply subtract the device key from the keyshare in the getter to make sure the modified keyshare is inserted at all right places. This is showed in figure 7.2.

Initially, our plan was to modify the keyshare directly in the database when it has to be altered. In the end we chose to add a new field in the database to store the device key separately from the keyshare. Therefore, the device key has to be injected in the keyshare getter. We did this, because a separately stored device key makes it easier to do error handling. The way we had it in mind first was to just overwrite the modified keyshare over

Figure 7.2.: Java code to calculate keyshare in keyshare server

```
public BigInteger getKeyshare() {
        BigInteger key = new BigInteger(getString(KEYSHARE_FIELD), 16);
        BigInteger delta = getDeviceKey();
        return key.subtract(delta);
}

public void setDeviceKey(BigInteger delta) {
        setString(DEVICE_KEY_FIELD, delta.toString(16));
        saveIt();
}

public BigInteger getDeviceKey() {
        return new BigInteger(getString(DEVICE_KEY_FIELD), 16);
}
```

the old keyshare. A potential problem then was that there might be issues when for instance an error occurs in the app during the restore process. The app might crash and forget the negotiated device key while the keyshare server already modified the keyshare irreversibly in the database. A potential risk is that an account becomes fully unusable at some point. When the keyshare is stored separately, always a new device key can be generated.

We also considered solving this issue by storing the device key in memory for some time after a change, but this does not solve the following, more fundamental problem. If it is stored temporarily it will be erased after some time. This causes that old backups (made before a certain recovery moment) cannot be used anymore, because in those backups device keys negotiated after the backup moment cannot be known. This means all old backups become invalid after a recovery moment. We think this is hard to explain to users.

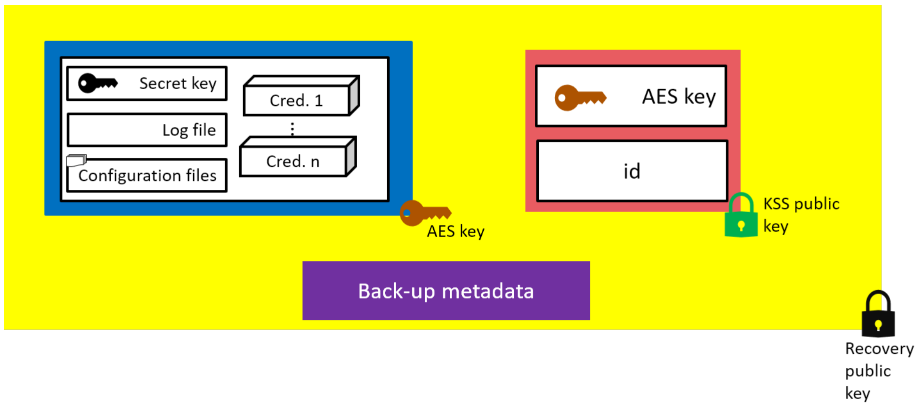
Just like the keyshare itself, the device key has a key length of 256 bits. Therefore all bits of the keyshare can potentially alter by the modification of subtracting the device key from the keyshare.

7.2. Constructing a Backup File

In section 6.1 we described that we are going to use a mnemonic phrase for users to store the decryption key in combination with a second authentication factor via trustees and/or a trusted party. We concluded that when the keyshare server is needed for other restore functionality anyway, this party can also serve as a trusted party for two-factor authentication.

Another issue we have to solve is that in some way we have to enforce that a user can only recover his backup in cooperation with the keyshare server (in his recovery role). Otherwise the keyshare server cannot enforce that old devices are invalidated according to the chosen device revocation strategy from section 6.2. This means the keyshare server will become a party that is involved in the restore process. Because of this and the conclusion about

Figure 7.3.: Backup outline



two-factor authentication we described above, we will use the keyshare server as a trusted party for two-factor authentication in our proof-of-concept.

To realize this, we came up with a backup model with multiple encryption layers to enforce that both the user and the keyshare server are needed to decrypt a backup. We designed this in such a way that the user is in control of the decryption via his key. In the end only the user (to be precise: the user's app) can access the credentials. We call this model the **matryoshka encryption model**. In figure 7.3 we visualized how this model looks like for IRMA backups.

Both the user and the keyshare server have a public-private key pair. The private key of the user is directly derived from the mnemonic phrase. The key pair of the server is just a regular public-private key pair. When making a backup, the user's IRMA app encrypts all user data (secret key, credentials, logs, etc.) with a freshly generated AES key. We call this packet the blue packet referring to the color of this packet in figure 7.3. The AES key is then combined with the username of the user and that combination is encrypted with the public key of the keyshare server. We call this packet the red packet. The red and the blue packet, combined with some metadata about the backup, form the actual backup file. The metadata is about which keyshare server is used, which salt (in IRMA called nonce) is used for the PIN, the user's username and the user's recovery public key that was used to encrypt the backup. The salt and the username are needed to contact and authenticate at the keyshare server. The user's recovery public key is stored to be able to make new backups after an earlier backup is restored. This is done to improve the usability of mnemonic phrases. Storing the public recovery key in the IRMA app and in backups makes sure a mnemonic phrase can be re-used over multiple backups.

The fact that we want it to be possible to use a mnemonic phrase for multiple backups is the reason why we chose for public key cryptography. In our design we encrypt the backups with the public key and restoring a backup is only possible having the private key. This means we can keep the public keys stored in a device such that backups can be made at every moment. The private key is converted to a mnemonic phrase and is erased from the device after the user wrote it down. This guarantees that attackers cannot access backups and cannot do a restore when having a compromised device.

To store the backup we first convert the relevant files in the IRMA app's internal storage to a JSON string. The files we include are all stored in the `/data/data/org.irmacard.cardemu/v2` folder of the app storage[1]. We store all files except for `irma_configuration`, because there is no need to backup the publicly known configuration files. We load the data from storage into the existing data structures of the IRMAGo library and then we convert those data structures to JSON using the default serializing features of Go. We now store the following data in a backup:

- The user's secret key
- The credentials belonging to the keyshare server the backup keys are configured for
- Paillier keys (if still present)
- Logs of performed disclosing, issuing sessions and signatures at moment of making the backup
- App preferences
- Metadata about all configuration updates
- Metadata about the keyshare server

In a backup we can only store the credentials that belong to the keyshare server that is involved in the particular recovery process. Credentials from other schemes do not belong to the control of that keyshare server, so it would not be neat if those credentials would be included in the backup. This means that when multiple keyshare servers would be involved, the user should also generate multiple backups.

A disadvantage for storing the logs in the backup is that not all changes in the log files might be stored. The log files can change on a daily basis, dependent on how much the IRMA app is used by the user. Because of the proof-of-concept character of the implementation we made, we did not further develop a solution for this problem.

7.2.1. Mnemonic Phrase

To implement mnemonic phrase generation we use a Go implementation of BIP39 made by Smith[2]. We use this library to convert a key as byte array to a representation in words and vice versa. Each word represents 11 bits of the byte array and the library produces as many words as needed for the length of the given byte array. The function `EntropyFromMnemonic` of this library takes a string with a mnemomic phrase (words separated by spaces) as an argument and outputs the key in bytes. The function `NewMnemonic` does the opposite.

We now only use the English word list. The library supports English, French, Italian, Spanish, Japanese, Korean and Chinese (simplified and traditional). It is relatively simple to add new languages to this library. The primary language of IRMA is Dutch and we found a project to automatically generate word lists for Dutch[3]. After a small manual check of the generated list, the list can be added and then the library also has support for Dutch. In the proof-of-concept we stuck to the English word list.

[1]A detailed overview of how the storage of the IRMA app is organized can be found in section 2.3.
[2]https://github.com/tyler-smith/go-bip39
[3]https://github.com/mccwdev/bip39-dutch-wordlist

Technical details As explained in the introduction of section 7.2, the mnemonic phrase is a representation of the user's recovery private key. Apart from which public key cryptography system should be used to generate this private key, a representation of private key should always be converted to words by the BIP39 library. The number of words that this library needs to represent the private key may not be too high, because this hurts the usability of the recovery mechanism. After all, the user is expected to physically write down these words. Converting a 2048-bit RSA private key naively would cost $\lceil 2048/11 \rceil = 187$ words and even a short key of 256 bits already takes $\lceil 256/11 \rceil = 24$ words. Because there hardly are public cryptography systems that securely provide keys shorter than 256 bit, we always need a key derivation function (KDF) to extend the mnemonic phrase to a proper key length.

In the proof-of-concept we used Percival's Scrypt key derivation function [23]. This KDF is designed to not only be secure against CPU brute-forcing, but also against parallelized set-ups like in GPUs or in dedicated hardware circuits used for proof-of-work in blockchain systems. In Go a Scrypt implementation is included in the standard library[4]. With respect to Scrypt's hardness configuration, the library's documentation recommended in 2017 to use the settings $N = 2^{15}, r = 8$ and $p = 1$. We have followed this recommendation, but because the recommendation is two years old we increased the main CPU/memory cost parameter to $N = 2^{16}$. It might be needed to lower this configuration if turns out that these settings slow down the key generation process on older devices too much.

In the proof-of-concept we used a 128 bit key as basis. Converted to a mnemonic phrase this will take $\lceil 128/11 \rceil = 12$ words. We consider this as a reasonable number of words to write down. Usability research has to be done about what number of words is acceptable in practice. Because of the strength of the KDF it would even be possible to further reduce the number of words if 12 words is still considered as too much.

7.2.2. Key Details

User's recovery key pair For the user's recovery key pair we use elliptic curve cryptography. We use curve25519 as curve and NaCl box as encryption technique. NaCl box uses Elliptic Curve Diffie-Hellman (ECDH) on curve25519 with Salsa20 as a stream cipher and Poly1305 for message authentication codes [2]. NaCl box can encrypt messages of arbitrary length.

We have chosen for elliptic curve cryptography here, because we need to generate a key pair based on the output of the KDF that was used to extend the mnemonic phrase. Curve25519 has the advantage that every possible bit sequence of the right length is a valid private key. NaCl box uses a key length of 256 bits. This means we can just let the KDF produce a 256 bit key and immediately use this as private key. The public key can directly be derived from this private key.

A disadvantage of ECDH is that by definition you always need two key pairs. User a's public key p_a and user b's private key s_b are combined to generate a symmetric key $ECDH(p_a, s_b)$. The symmetric key is then used for the encryption of messages between user a and b. During decryption the same symmetric key can be found using s_a and p_b. This means message authentication is included by design. In our use case, we do not need

[4]https://godoc.org/golang.org/x/crypto/scrypt

message authentication, because the public key is not really public. It is just known to that particular instance of the IRMA app. Adding authentication using a second key does not guarantee that the backup is authentic, because when a device gets compromised the authentication private key in the device is compromised too. This means one public key pair is sufficient. This is not possible in ECDH.

The other option we had was using RSA instead. A property of RSA is that not every possible bit sequence of the right length is a valid RSA key. The key has to meet some requirements. This means that a key generation process is always needed. The key that is produced by the KDF should then be used as a seed to that key generation process. We considered using RSA instead, since we already need RSA in other parts of the implementation. This would reduce the number of dependencies. However, we did not find a library in Go that has a full implementation of RSA key generation based on derived keys. Therefore we maintained the elliptic curve implementation.

To solve the issue of needing two key pairs in ECDH, we used a fixed zero key as authentication private key and the corresponding public key as authentication key pair. The key pair derived from the mnemonic phrase we used as encryption key pair. It is important to do it in this order. Like in most public key cryptography systems, in elliptic curve cryptography the public key can directly be derived from the private key. It is therefore important that the actual key pair is not used as authentication key, because then the public key has to be used as decryption key. Given the earlier mentioned property of being able to generate the public key having the private key, the private key should only be stored in the mnemonic phrase and nowhere else.

Since we need two rounds of encryption for our matryoshka scheme we as well examined whether we could combine these two rounds into one ECDH encryption. It could have been possible to use the authentication key pair from ECDH for the encryption of the keyshare server encryption layer. However, this turned out to be impossible, because ECDH does not allow us to use two private keys. It depends on the use of one public key and one private key. Otherwise the Diffie-Hellman shared key cannot be found anymore in decryption. Therefore we stick with the approach of using a fixed authentication key in the ECDH encryption process and using a separate encryption layer for the keyshare server's encryption.

Keyshare server's recovery key The keyshare server also needs a public key pair for the encryption of the red packet containing the AES key of the particular backup. The IRMA app should be able to encrypt the backup using the public key of the server and the keyshare server should be able to decrypt it. Therefore we need an encryption mechanism that both supports Go for IRMAGo and Java for the keyshare server.

Since the red packet only contains an AES key, it is no problem that the keyshare server decrypts this packet directly. This design makes it possible to only have one single server key pair that is used for all users. This key can be generated during configuration of the keyshare server, so we can use practically every public key cryptography system.

Because elliptic curve cryptography had the disadvantage of needing two keys and it was hard to find a proper Java implementation of ECDH on curve25519 with Salsa20 and Poly1305, we decided to just use RSA. RSA is supported by many languages and there are many standards making it easy to use it cross platform.

We stored the recovery keys in `irma_configuration` in the main directory of the scheme where recovery is configured for. For example when a keyshare server is configured to serve the `pbdf` scheme, the keys are stored in `irma_configuration/pbdf/` as `recovery_private_key.der` and `recovery_public_key.der`. Obviously the private key should only present at the keyshare server and not in the public version of `irma_configuration`. In the proof-of-concept we used 2048-bit RSA keys, but in principle any size RSA key can be put in those files. The RSA library for Go has no default support for reading keys from DER or PEM files. For now to make it work, we manually converted the public key to a file containing the RSA modulus in decimal form. We stored this as `recovery_public_key.key`.

AES key The blue packet of the backup is encrypted with a fresly generated 256 bit AES key. The app performs both the encryption and decryption itself such that other parties cannot learn any of the information from the backup.

7.3. Changes to the Protocols

In order to realize the recovery process, we added some new protocols for the restore process and the backup making process. Furthermore, we designed a protocol for the initialization of the new keys needed for recovery and the needed registration of those keys at the keyshare server.

To realize device revocation via the keyshare server, the current protocols for disclosure, issuance and signing do not have to be changed. The only changes needed are in the calculations of the proof as described in section 7.1. This is the same for every proof, so it does not require a protocol change.

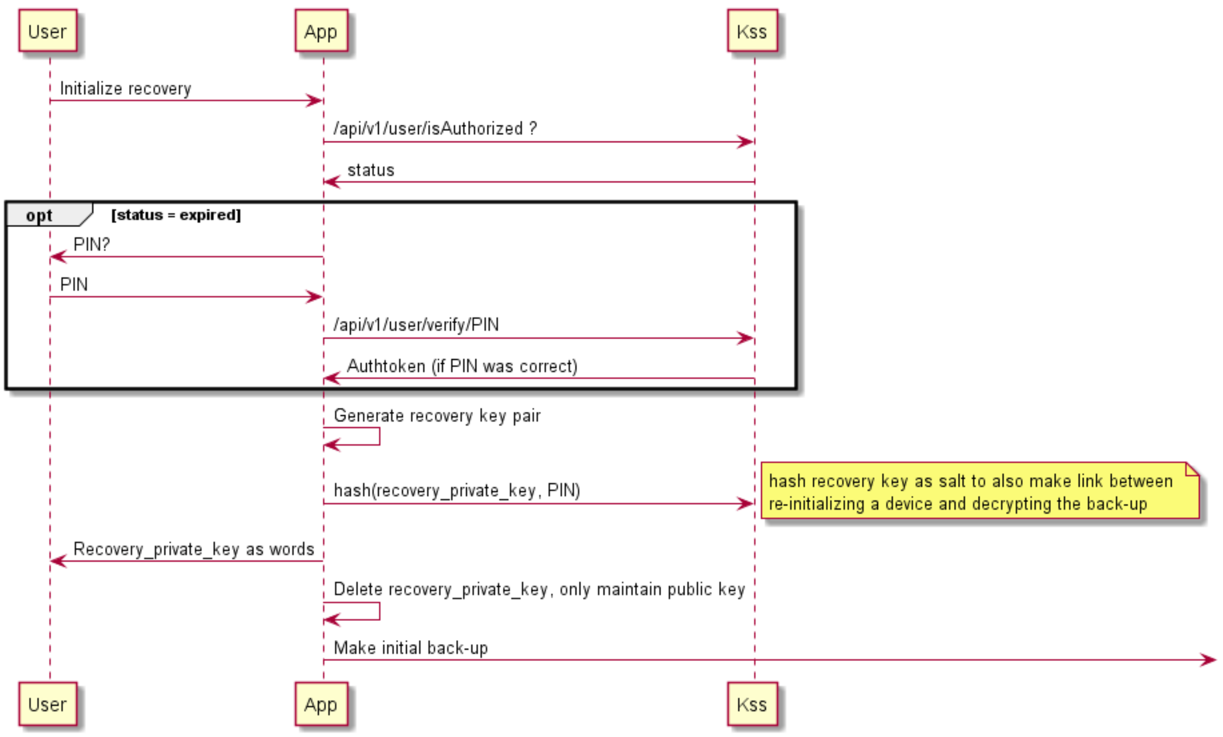
7.3.1. Recovery Initialization

At first keys have to be generated in order to be able to encrypt backups. Because the restore process is a cooperation of the user's IRMA app and the keyshare server, the key initialization process also requires some communication between both parties. We had to design a new protocol for this communication, because in the baseline implementation of IRMA no recovery keys exist yet. This means existing users first have to set up recovery. Also new users should write down their recovery key. Since this can only be done in a trusted environment, this is not a step that can be enforced during enrollment. For these two reasons we made a separate protocol.

In case of the user's recovery key, the keyshare server is not involved. This key can be generated without needing a communication protocol. The only requirement is that the private key should be converted to words and is displayed to the user. The user should take the time to write down the mnemonic phrase. The public key of the user recovery key should be stored in the IRMA app locally. The private key should be deleted after the words are written down by the user.

For the second phase of initialization we do need a communication protocol. At the keyshare server a new hashed PIN for recovery must be uploaded. We do not want to introduce a second PIN code the user has to memorize, so the app should enforce that the recovery PIN is the same as the regular PIN. The only difference between both PINs is that the recovery PIN is hashed with a fresh salt. We have done this, because the regular PIN

Figure 7.4.: Initialization recovery


User
App
Kss
Initialize recovery
/api/v1/user/isAuthorized ?
status
opt
[status = expired]
PIN?
PIN
/api/v1/user/verify/PIN
Authtoken (if PIN was correct)
Generate recovery key pair
hash(recovery_private_key, PIN)
hash recovery key as salt to also make link between re-initializing a device and decrypting the back-up
Recovery_private_key as words
Delete recovery_private_key, only maintain public key
Make initial back-up
User
App
Kss

salt is stored in the IRMA app. This means that the salt might be compromised when a device gets lost. This is not directly a problem for security, since the PIN check has still to be done by the keyshare server. The keyshare server can therefore detect potential brute-force attacks. Refreshing of the salt is done to prevent that an attacker can find the value of the user's PIN code when hashed PIN codes get breached at the keyshare server. Another advantage is that when multiple keyshare servers are used, even when they gang up on the user, they cannot determine whether that user uses the same PIN at multiple servers.

The protocol starts with a regular authentication session to verify the user entered the right PIN code. This is both for the keyshare server to check that the right user is configuring recovery and for the app to check the entered PIN was correct. When authentication succeeds, the app sends the new recovery PIN hash (based on a new salt) to the keyshare server and the keyshare server will store that. The new protocol is visualized in figure 7.4.

7.3.2. Making a Backup

We designed the construction of backup files in such a way that no interactive protocol with the keyshare server or any other server is necessary. How a backup is structured is explained in section 7.2.

When a backup file is generated, some communication might be needed to export the file to the desired storage place. As discussed in section 6.1, we did not focus on finding convenient storage locations for backups in this research. Nevertheless, to make the proof-of-concept work we have to somehow export the backup file from the IRMA app in order to transfer it to other devices. In the proof-of-concept we solved this by using existing techniques from the IRMA app for exporting files. For this no new communication protocol was needed. It was largely a user-interface issue for the app. Therefore we discuss the details of exporting backups at the app implementation explanation in section 7.4.2.

7.3.3. Restoring a Backup

For restoring a backup we do need a communication protocol, because also the keyshare server is involved in this process. The order of messages is important, because we want to guarantee that the keyshare server does not reveal the AES key of a certain backup until it is certain the user is properly authenticated and the IRMA app on the user's old device is invalidated.

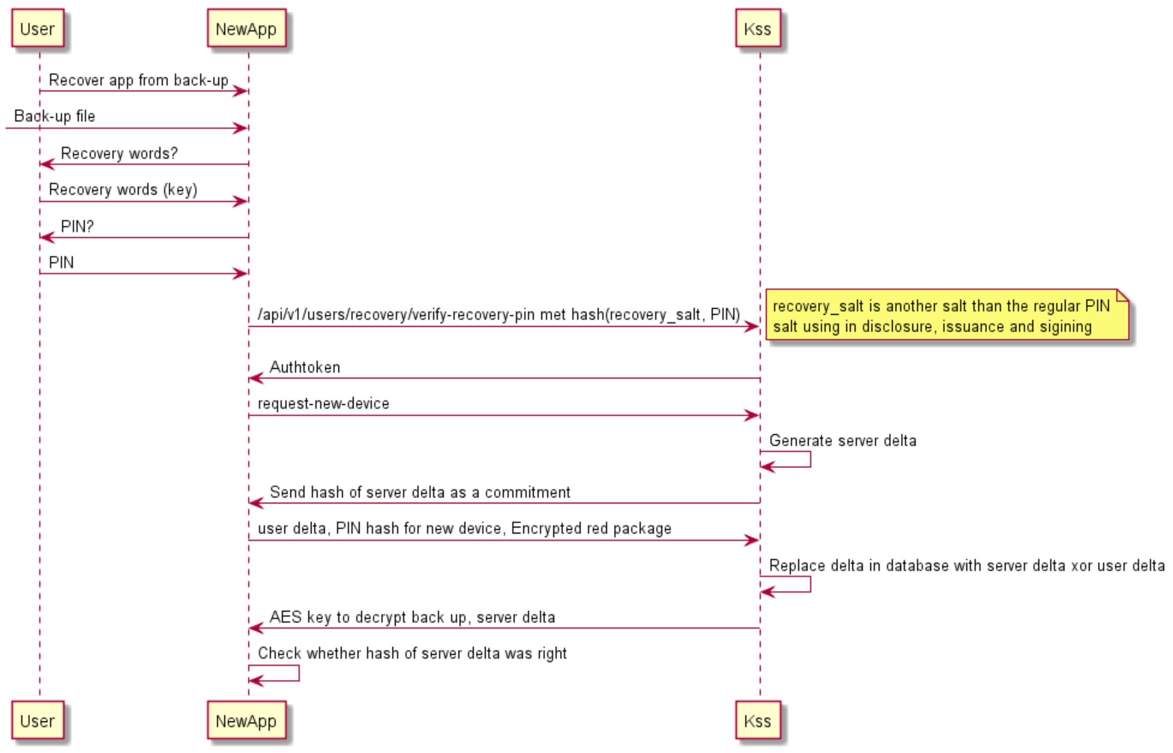
Before the process is started, the backup file should be loaded in the app and the user should have entered his mnemonic phase. In this way the app can derive the user's recovery private key and decrypt the first layer of encryption without the need of any communication protocol. As described in section 7.2 the backup metadata can already be accessed when the first layer of encryption is removed. The metadata contains the information that is needed for certain messages of this restore protocol.

In the communication protocol first the user has to authenticate to the keyshare server using the recovery PIN hash. The salt of this PIN hash is stored in the metadata of the backup. When this is completed, the user can do a request for a new device at the keyshare server. The keyshare server will then generate randomness for a new delta on the secret key (so for the new device key). It will not send the random number to the user yet to prevent that the user is able to choose his randomness to get the device key of his choice. Instead, a hash of the randomness is sent to the user as a commitment on the randomness. When the commitment is received, the user generates his part of the randomness for the device key. The random number is sent to the keyshare server together with the encrypted red packet (see section 7.2) and a new PIN code hash for the new device. The keyshare server must check whether the user id included in the red packet corresponds to the user id of the authenticated user. This to prevent that the keyshare server is used as a decryption oracle. If this happens to be correct, the keyshare server combines the randomness of the user and the randomness of the keyshare server and stores the new device key in the database. It also stores the new PIN hash. When everything is stored, the decrypted red packet is sent to the user. In the final message also the server's randomness for the device key is included such that the user can also calculate the new device key. The user has to check that the keyshare server's random number corresponds to the commitment it received earlier to prevent that the keyshare server is able to fully determine the value of the device key itself. With the AES key the user received, the user can decrypt the blue packet of the backup and recover all the data. This protocol is visualized in figure 7.5.

7.4. App Implementation

In order to get a working proof-of-concept for the recovery features that we have implemented in IRMAGo, we added functionality to the IRMA app to be able to make backups and restore a backup using the actual interface. We therefore had to design three new user

Figure 7.5.: Perform recovery



User
NewApp
Kss
Recover app from back-up
Back-up file
Recovery words?
Recovery words (key)
PIN?
PIN
/api/v1/users/recovery/verify-recovery-pin met hash(recovery_salt, PIN)
recovery_salt is another salt than the regular PIN salt using in disclosure, issuance and sigining
Authtoken
request-new-device
Generate server delta
Send hash of server delta as a commitment
user delta, PIN hash for new device, Encrypted red package
Replace delta in database with server delta xor user delta
AES key to decrypt back up, server delta
Check whether hash of server delta was right
User
NewApp
Kss

flows in the app: one for initializing recovery and setting up mnemonic phrase, one for making a backup and one for restoring a backup.

7.4.1. Recovery Initialization

Before recovery can be used, at first key material should be generated by the user. This cannot be done in the background, because the user should configure a mnemonic phrase. It could be generated automatically, but the user must write the phrase down immediately after it was generated. After all, the phrase must be erased from the device after initializing, because it is a representation of the user's recovery private key. Furthermore we need the user's PIN code, because this is needed to fulfill the initialization protocol we described in section 7.3.1.

This means the initializing process has the following steps:

1. When a user wants to generate a new backup and recovery is not configured yet, the user is asked to configure recovery. If the user continues, the process is started. Since keys have to be generated this can take some time.

2. The mnemonic phrase is displayed to the user. The user has to write the words down that are displayed. Ideally, this is done on paper, because we assume most people intuitively understand how to protect a paper with a secret. We will include this recommendation in the explanation to the user on the screen.

3. The user has to confirm he wrote down the words. This to emphasize to the user that there is no way how this particular phrase can be obtained again later. It is possible to generate a new phrase and make new backups if the key is lost. Once a phrase is lost, all backups encrypted with that phrase are useless.
4. The user has to enter his PIN as a confirmation of the process.

Instead of physically writing down a mnemonic phrase, users might also take the easy road and make a screen shot of the mnemonic phrase when it is displayed and store this. We expect that most users do not sufficiently know how to securely store a screen shot. We did not consider this vulnerability in the design, but it emerged to us when making the proof-of-concept implementation. For now we did not implement any measure to prevent this user behavior, but when realizing recovery in production a measure should be considered.

Screen shots of the IRMA app corresponding to every step in the process can be found in figure 7.6.

7.4.2. Making a Backup

In the design of the proof-of-concept we decided to keep the export possibilities of backups simple for now. Exporting backups to files is hard, because especially iOS has no clear concept of files. Furthermore, implementing a feature for sharing the backup with for example cloud solutions (Google Drive, Dropbox, etc.) may require a lot of time, because we have to find out how the APIs of those applications work and how they can be integrated in React-Native. Therefore we decided to use e-mail as export mechanism in the proof-of-concept. This feature is already available in the app to send signature files belonging to IRMA signature sessions. Since attributes currently only have plain text as data type, backups are only a few kilobytes in size. This can be sent over e-mail easily.

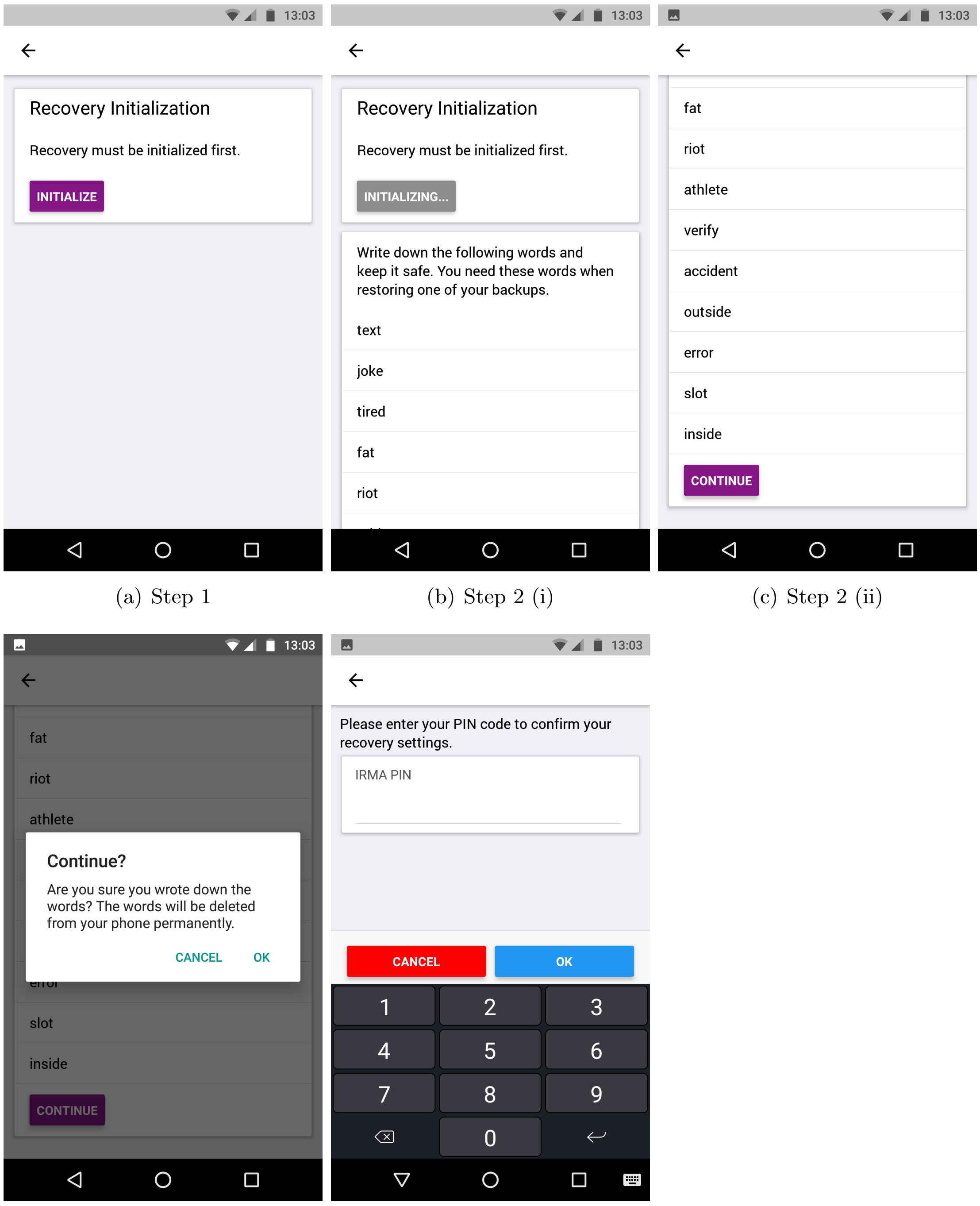
For the user interaction this means that the user flow in app is basic. It will just be a button to activate the backup making process. The application should compose and encrypt a backup file and then launch the e-mail application. The user should manually send the backup file to his own mail address.

During the implementation we found out that to use IRMA's existing e-mail functionality it is practical to pass the destination e-mail address to the mail client. We therefore added a text field in the user-interface for making backups where the user can specify his e-mail address. By default we fill in one of the email addresses the user possesses as an attribute. If there are no instances of the email credential available, the user has to fill in the address himself.

Another finding we did is that support of multiple keyshare servers is not fully supported by the version of the IRMAGo library we used. We did try to implement support for generating multiple backups when multiple keyshare servers are in place, but due to the missing support from other parts of the library this was not always possible. Therefore we had to make the assumption only one scheme is in place that uses a keyshare server. In the current architecture this happens to be the case, so this forms no restriction at the moment.

We did not implement automatic backup generation. Instead we added a item in the app's menu drawer to manually start the backup process. Some screen shots of how the user interaction looks like can be found in figure 7.7.

Figure 7.6.: App screen shots of recovery initialization


13:03

Recovery Initialization

Recovery must be initialized first.

INITIALIZE

(a) Step 1

13:03

Recovery Initialization

Recovery must be initialized first.

INITIALIZING...

Write down the following words and keep it safe. You need these words when restoring one of your backups.

text
joke
tired
fat
riot

(b) Step 2 (i)

13:03

fat
riot
athlete
verify
accident
outside
error
slot
inside

CONTINUE

(c) Step 2 (ii)

13:03

fat
riot
athlete

Continue?

Are you sure you wrote down the words? The words will be deleted from your phone permanently.

CANCEL OK

slot
inside

CONTINUE

(d) Step 3

13:03

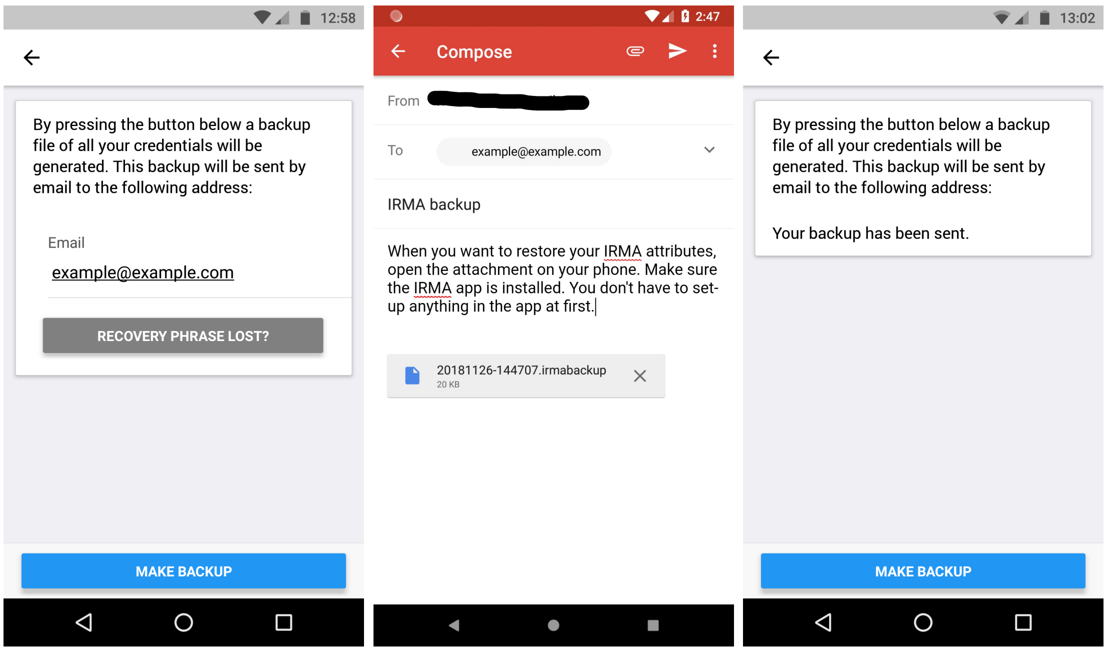
Please enter your PIN code to confirm your recovery settings.

IRMA PIN

CANCEL OK

(e) Step 4

Figure 7.7.: App screen shots of backup making process


12:58

By pressing the button below a backup file of all your credentials will be generated. This backup will be sent by email to the following address:

Email
example@example.com

RECOVERY PHRASE LOST?

MAKE BACKUP

2:47
Compose

From [redacted]

To example@example.com

IRMA backup

When you want to restore your IRMA attributes, open the attachment on your phone. Make sure the IRMA app is installed. You don't have to set-up anything in the app at first.

20181126-144707.irmabackup
20 KB

13:02

By pressing the button below a backup file of all your credentials will be generated. This backup will be sent by email to the following address:

Your backup has been sent.

MAKE BACKUP

(a) Generate (b) Mail (c) Done

7.4.3. Restoring a Backup

When a user freshly installs the IRMA app, an option is displayed to load your previous IRMA account by supplying a backup. We pictured this as a 'Recover' button right next to the 'Open account' button on the start screen. When choosing to load a backup the user should give his backup file to the app. After that, the user has to enter his mnemonic phrase. If correct, the user must enter his PIN code. When this is completed, the backup can be decrypted and the credentials can be recovered in the IRMA app.

During implementation we found out that it is hard to make a clear, cross-platform usable interface to let users import a backup file in the IRMA app. Especially on iOS this is hard, because loading files is only implemented for images there (via the 'Camera Roll'). Therefore we decided to change the flow a bit. We made a file association between the IRMA app and `.irmabackup` files such that backups can be opened from the e-mail application directly. When pressing on a backup file in the attachment of an e-mail, the IRMA app starts and begins with the recovery process. We still added the 'Recover' button in the start-up screen. This now opens a screen with explanation how a backup can be loaded. The new start screen can be found in figure 7.8a and the explanation screen can be found in figure 7.8b. The user flow is implemented as follows:

1. The user opens a backup file with the IRMA app.
2. The IRMA app asks the user to enter his mnemonic phrase.
3. The user enters his mnemonic phrase and presses 'Continue'.

4. The IRMA app checks whether the key is correct. If so, it decrypts the backup, loads the keyshare server metadata and asks the user to enter his PIN.

5. The user enters his PIN and presses 'Continue'

6. The IRMA app starts recovery process at the keyshare server, obtains all necessary information (decryption key, new device key, etc.) and decrypts the backup. After the process is completed, this is notified to the user and the credential dashboard is shown with all recovered credentials.

The screen shots of this flow can be found in figure 7.8. The numbers that are mentioned by the screen shots correspond to the step numbers from the user flow above.

At the end we found out that it is hard to enforce that a certain file is opened with a particular app. The Android universal link API, which is used when opening a file, does not support file extensions. Therefore we could not associate `.irmabackup` files with the IRMA app. As a solution we linked every file type to the IRMA app. This, however, is not a very neat solution, because Android now thinks any type of file can be opened by the IRMA app. For a proof-of-concept this is not a problem, so we kept it in this way. In a release version a work around should be found for this.

Source-code repositories of the implementation we made Below the URLs of the Git source-code repositories of the proof-of-concept are displayed. These links refer to the state of these repositories at the moment of writing.

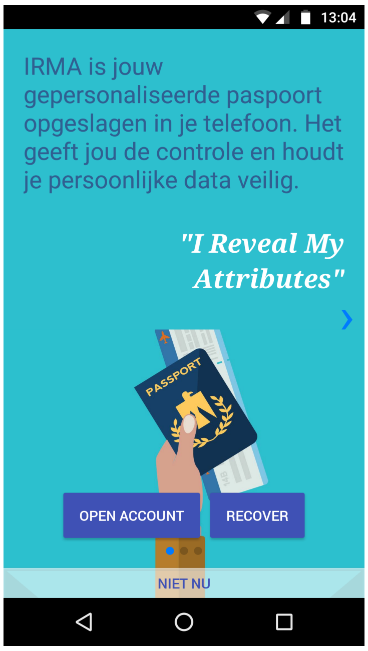
- IRMA keyshare server. `https://github.com/ivard/irma_keyshare_server/tree/ac0d58df3a1663a5ce3708e350c0758485a4133b`

- IRMAGo library. `https://github.com/ivard/irmago/tree/e6809ae0e69ed1d70f1e6a78b072e024e1a916e7`

- IRMA Mobile (code of the app). `https://github.com/ivard/irma_mobile/tree/602b1f968af2f6d5ec0c4f40620c14a3be4f9a6a`

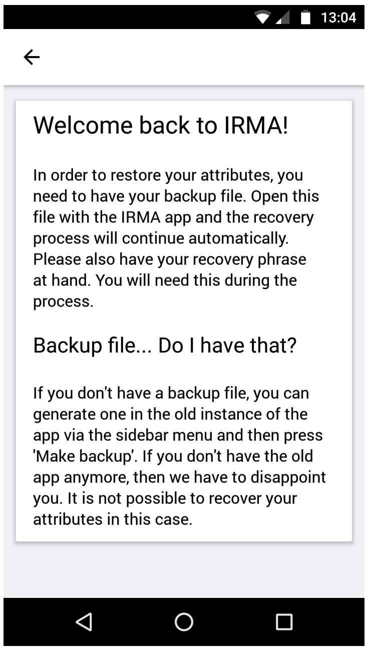
- Demo scheme with the configuration files that we used in our proof-of-concept. These configuration files should be put in place of the `irma_configuration/pbdf` directory to make the proof-of-concept work. `https://github.com/ivard/pbdf-schememanager/tree/9a6ab7a08f1d562b477fca7cbbc82d51e2a724f1`

Figure 7.8.: App screen shots of recovery process


IRMA is jouw gepersonaliseerde paspoort opgeslagen in je telefoon. Het geeft jou de controle en houdt je persoonlijke data veilig.

"I Reveal My Attributes"

(a) Start-up screen

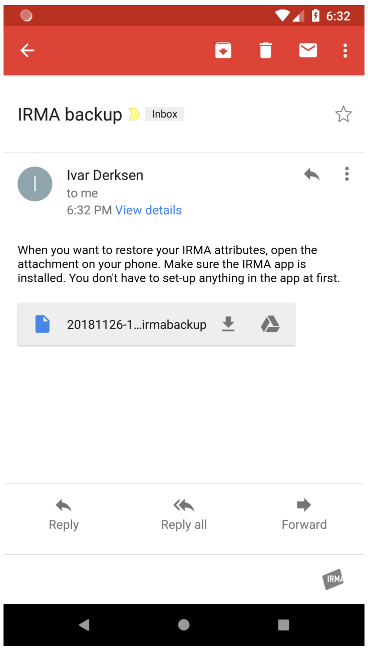

Welcome back to IRMA!

In order to restore your attributes, you need to have your backup file. Open this file with the IRMA app and the recovery process will continue automatically. Please also have your recovery phrase at hand. You will need this during the process.

Backup file... Do I have that?

If you don't have a backup file, you can generate one in the old instance of the app via the sidebar menu and then press 'Make backup'. If you don't have the old app anymore, then we have to disappoint you. It is not possible to recover your attributes in this case.

(b) Explanation screen

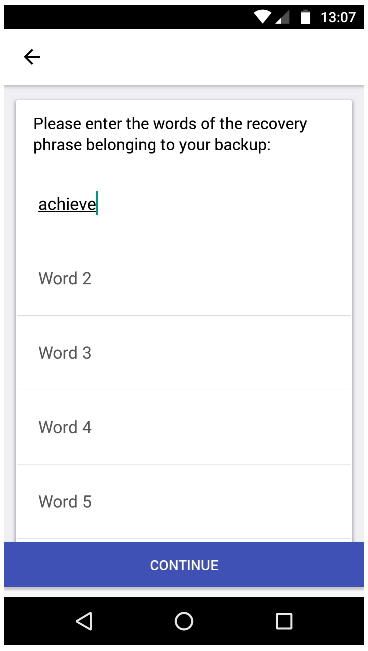

IRMA backup Inbox

Ivar Derksen
to me
6:32 PM View details

When you want to restore your IRMA attributes, open the attachment on your phone. Make sure the IRMA app is installed. You don't have to set-up anything in the app at first.

20181126-1…irmabackup

(c) Step 1


Please enter the words of the recovery phrase belonging to your backup:

achieve

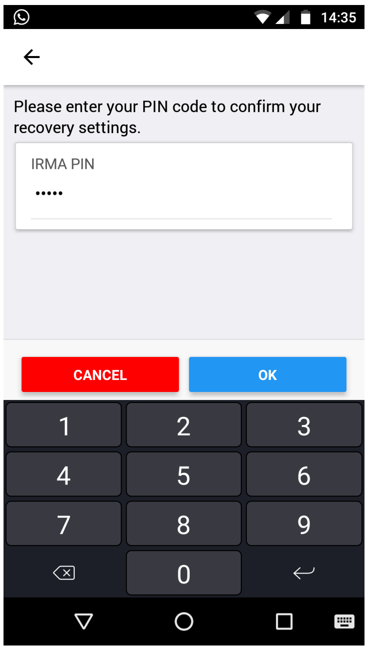
Word 2

Word 3

Word 4

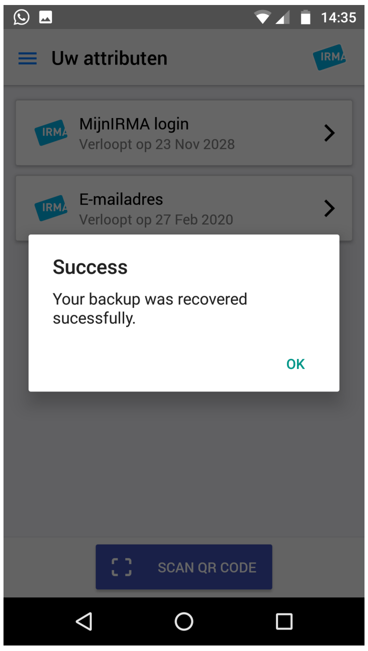
Word 5

(d) Step 2 and 3


Please enter your PIN code to confirm your recovery settings.

IRMA PIN
•••••

(e) Step 5


Uw attributen

MijnIRMA login
Verloopt op 23 Nov 2028

E-mailadres
Verloopt op 27 Feb 2020

Success

Your backup was recovered sucessfully.

(f) Step 6

8. Conclusions

In this research we contributed to identity platform IRMA by examining how recovery functionality can be introduced. We will discuss the conclusions in two sections. In section 8.1 we discuss what design is needed for recovery to fulfill the design criteria we made. In section 8.2 we discuss the contributions that were needed to implement the proof-of-concept.

8.1. Design of the Solution

Backup and restore We found out that recovery functionality for IRMA should be a backup and restore solution. This is needed to make sure all credentials can be recovered in all possible scenarios. A backup has to be stored somewhere and we cannot expect from all users that they exactly know how to properly protect it on the location where it is stored. We do also not want to impose a storage location to the user, because this harms portability. Therefore a backup should be encrypted in such a way it can be stored anywhere. The user's data may never leave the protected environment of the IRMA app without proper encryption.

In our design we came up with a solution to realize encryption of the backup in an understandable and usable way for the user that also gives control to the user. A challenge in this is key management. In this research we made a comparison of key management solutions for the decryption key. Our conclusion is that the best solution is a combination of two solutions. As primary solution we consider that the best option is a mnemonic phrase that can be written down on paper. No technical knowledge is needed to safely store away a piece of paper, so it is transparent and understandable for users. Additionally, also parts of the key must be managed by trustees of the user or a trusted institution as second authentication factor. This in order to prevent that the backup can be decrypted when the mnemonic phrase gets stolen.

Device revocation To guarantee control and existence we concluded that it is necessary for users to be able to block a device in case it gets lost. We call this device revocation. Theoretically, backups would even make it possible to use one IRMA account on multiple devices at the same time. This would however require all kinds of extra measures to make sure this cannot be exploited for identity fraud. Support of IRMA accounts on multiple devices at the same time is not within the scope of this research, so we chose to technically enforce that only one device can be used at the same time. In other words, device revocation should be mandatory in each restore procedure.

In this research we examined what options IRMA has for device revocation. We found some promising solutions that can be used. The option that we think is the best is having trusted parties that are responsible for device revocation. In this way there a single point of contact for users which makes the process transparent and understandable. Ideally, this trusted party issues its own IRMA credentials stating that a particular device is valid. We call these device credentials. All techniques for revocation we came up with (several techniques for credential revocation and revocation via a central party) can be used and verifiers have the freedom to ask for a device credential of a party they trust.

IRMA uses the keyshare server to check the user's PIN code. Letting a trusted party check the PIN is at the moment the only secure method that is also technically realizable. Since this central party is already involved in all IRMA sessions, this party can also serve as trusted party for device revocation. Device credentials are therefore not needed yet. We found a way of modifying the user's keyshare (stored at the keyshare server) with a device key. We make sure only one device at the time is able to know the device key and in this way we technically enforce only one device can be used at the same time. The device key is a symmetric key between the user's IRMA app and the keyshare server. This key is applied in such a way that the electronic signature on credentials, in which key material is included, remains the same. This does not depend on the value of the device key.

Another advantage of using the keyshare server is that it already uses a PIN code to authenticate users. This means we can also use the keyshare server as an authority for two-factor authentication on the backup. The restore process should be designed in such a way that it can only be finished when the keyshare server cooperates. The user benefits from this, because it makes sure that a stolen mnemonic phrase becomes revocable. The IRMA ecosystem also benefits from this, because the keyshare server helps guaranteeing existence by enforcing device revocation.

8.2. Proof-of-concept

In the proof-of-concept we made for recovery functionality in IRMA, we implemented the processes according to the design as described above. An important contribution of this proof-of-concept to the design of backups is its encryption model.

We introduce the matryoshka encryption model in which we use two encryption layers. Two layers are needed to enforce participation of the keyshare server in the restore process. When we would only use the user's mnemonic phrase to encrypt the backup, it can already be decrypted without the involvement of any other party. Then the participation of the keyshare server cannot be enforced. By using two layers of encryption we can both use the user's mnemonic phrase to enforce control and a key of the keyshare server such that the keyshare server must be involved to decrypt a backup. The outer encryption layer is based on the user's mnemonic phrase, such that the keyshare server cannot perform any decryption step without involvement of the user. Public key cryptography is used such that it is not needed to configure a new mnemonic phrase for every backup.

9. Future Work

We will discuss this research's future work in two categories. At first we will discuss the elements of future work for realizing recovery in IRMA. Especially converting the proof-of-concept to a solution that is ready for production requires some work. After that we will discuss some elements that are missing in IRMA according to Alliander, besides recovery.

9.1. Future Work in Recovery

In our research we presented a proof-of-concept for recovery functionality in IRMA. The programming code of the proof-of-concept is not suitable to use in production yet. To realize this the following steps are still required.

- **Improve stability of the code**. The core functionality for recovery in the proof-of-concept is working. However, we did not pay a lot of attention to error handling and maintainability during the design. In the regular flows the code is working fine and the most errors also trigger error messages. However, the error messages do not all give a clear message to the user what went wrong. Sometimes just the raw error message is printed that was raised by some internal library. Furthermore, sometimes the errors behave annoyingly. For example, some error messages fully abort the recovery process and the user should then start from the beginning again. This should be improved. Finally, the IRMA app now recognizes every file as a possible backup file. A solution has to be found to make sure that only `.irmabackup` files are associated with the IRMA app.
- **Check encryption algorithms** In the proof-of-concept we used several cryptographic systems to encrypt the backup. We now use AES for the encryption of the backup itself, RSA for encrypting the keyshare server's part containing the AES key and Elliptic Curve Diffie-Hellman for the user's encryption layer. Especially this last one is not fully ideal for its use case, since ECDH only supports authenticated encryption. Authenticated encryption is not needed for this use case. A more detailed explanation of this issue is described in the part about the user's recovery key pair in subsection 7.2.2.

  It might be possible to replace ECDH by RSA. Then a key generation library for Go must be found to convert the output of the key derivation function to a RSA key pair. Another option is that another cryptographic algorithm must be found that supports encryption using only one key pair. This algorithm is ideally used for the server part of the encryption too, such that we do not need to include multiple cryptographic libraries. This saves some dependencies.
- **Make the texts in the app more user-friendly** In the UI of the proof-of-concept app we used hard coded strings for explanation texts, texts in buttons, etc. The IRMA

app's source code knows special files where all displayable strings are recorded. The strings for recovery should be moved to these files. This means all these strings should also be translated to Dutch for the Dutch string file. Furthermore, the texts we made are written to be functional. We tried to explain what a user has to do, but we did not examine the clarity and understandability of these texts. Ideally, attention is paid to the clarity of the texts directly when they are translated.

- **Mnemonic phrase in Dutch** The words for the mnemonic phrase can only be generated, using English words in the proof-of-concept. In the library we use, no language file for Dutch is provided. We found a tool that is able to generate such a list for Dutch. We explained this in section 7.2.1. The list that this tool generates should be checked manually whether all words are acceptable. For example, similar words, difficult words or words with a strange spelling should be replaced. When having this file, phrases can also be generated in Dutch.

- **Auto-complete the input for a mnemonic phrase** Since all words of the mnemonic phrase are in a fixed list, the app could help the user with some error correction when he is typing. The app can for example suggest a word during typing or warn that a word is not in the list. Since the first four characters of a word are unique, it is already possible to suggest the word after the user has entered four characters. This saves the user a lot of effort.

- **Let people confirm the phrase** A nice to have additional check during initialization is that the user is asked to confirm the mnemonic phrase before it is deleted by the app. In this way the user is forced to check whether he wrote down the phrase correctly. When this is not done, there is a risk that users only find out they wrote down the wrong key when they are trying to restore a backup. Then it is already too late. This confirmation step prevents these potential problems. When this feature is added, research most be done about how many words a mnemonic phrase may contain to remain user-friendly.

- **Prevent screen shots of mnemonic phrases** As discussed in section 7.4, it is likely that users are going to make a screen shot of their mnemonic phrase instead of physically writing it down on paper. Research must be done whether it is sufficient to only recommend users to write the phrase down on paper. When it turns out that this is not sufficient, measures should be taken to prevent users from doing this.

- **Location to store backups** In this research we abstracted from where the backup should be stored. In the proof-of-concept we used e-mail for this, because this was the easiest to implement. Some research must be done into how backups can be stored best. In our design backups are fully encrypted, so it does not matter at which party they are stored. The only thing that matters is that the backup location should be persistent and user-friendly. User-friendly in this context means that ideally the backup process is done automatically without the user having to worry about it.

  For example backups could be stored in Google Drive or iCloud, a bit similar to how WhatsApp does its backups. In this way you could automatically make backups and upload those to external locations.

Another possibility might be to add backup storage as an additional service of the keyshare server. The keyshare server can even store a backup in a more structured way. For example, you could then make a backup per credential instead of only allowing backups with credentials grouped as one package. In this way you can make sure that all credentials are immediately backupped and even that expired credentials are automatically deleted. A disadvantage of this approach is that encrypted personal data is considered as personal data according to the GDPR. This makes this solution legally difficult.

- **Storing logs separately from the backup in case of manual backups** A bit related to the previous problem is storing the log files. In our proof-of-concept the log files are included in the backup. Whereas the set of credentials does not change very much over time, the log files may change on a daily basis. This means that to make sure the logs are properly preserved, backups have to be made more frequently. When backups can be made automatically, this is not really a problem. However, when the solution for the previous problem is a solution where backups have to be stored manually by the user, it it important that at least the log files are stored automatically.

Besides the improvements that have to be done to the proof-of-concept still, we also found a more fundamental issue. A possible attack that already exists in IRMA is that users can cherry-pick an identity together by gaining derived credentials, then deleting the base credential that was used to gain the derived credential and finally getting issued new base credentials. For example it is possible to gain an over-18 attribute, delete the identity of the original person (name, date of birth, etc.) and retrieve the identity of another person. Now you can potentially disclose over-18 combined with identity details of a minor.

Currently this is prevented by making it impossible to transfer the app to another device and making sure the storage of the IRMA app can only be accessed by the IRMA app itself. backups will break this security mechanism. Users can now give backups containing credentials to other users and they can load these backups in their device. backups can still only be loaded in one device at the same time (due to device revocation), but the user can just make new IRMA accounts over and over again.

It is important that this attack vector is mitigated. In the current situation, this attack is already exploitable when using a rooted device. backups only make this attack more visible to users. Therefore it is wise to look for a solution for this problem, even when backups are not implemented in IRMA. We came up with three possible options for this.

- The easiest solution is to only allow non-identifying attributes to be combined with identifying attributes in a credential. In this way, it is not possible for users to delete all identifying attributes from the app without also deleting all non-identifying attributes. Verifiers can check whether the identity is right by also demanding that the identifying attributes from the credential are disclosed.

  From privacy perspective, it is of course not desired in some scenario that identifying information is disclosed. In these scenarios, another option might be possible. Users can prove that the value of an attribute from a reliable credential corresponds to the identifying attribute of a credential with non-identifying attributes. The value of the identifying attribute itself does not have to be disclosed. The user only has to prove that the values correspond. In this way a threshold is raised for users to work along with an attack, because they also have to give their identifying attributes to the attacker to make the attack work.

- During issuance the user can show to the issuer a pseudonym of his secret key and the issuer's public key. This does not leak any information, since the pseudonym will be different at other issuers and the pseudonym is not used during disclosure. This means there is no linkability among issuers and also not between issuing and disclosing sessions. The issuer can store what person belongs to which pseudonym. When the attack is performed, at some time the attacker will come back at that issuer requesting another identity for the same pseudonym. When this happens, it can be identified by the issuer and then the attack can be prevented.
- In credentials an extra attribute can be added to link derived credentials. For base credentials the value of this attribute is randomly generated. However, for derived credentials the value of the parent is used. When a credential is dependent on more than one credential these values must be combined in some way. This should be done in such a way that verifiers can check whether the value of this extra attribute in a derived credential corresponds with the value of the corresponding base credentials. A potential problem of this approach is that a base credential cannot be replaced, because the new credential's linking attribute gets a new random value. In this case the value of the derived credential's linking attribute is not right anymore. A work-around has to be found for this.

To further mitigate the risk of this attack, it could additionally be an idea to encrypt credentials in app storage. The IRMA app is recently extended with a feature that requires users to give in their PIN code to unlock their IRMA app. In this step also a decryption step of the storage can be added. The decryption key can be managed by the keyshare server. This scenario is particularly interesting for rooted devices, since the storage of those devices is not protected. This means other apps might be able to access the data and leak it. For non-rooted devices the current protection is already sufficient.

9.2. Other Desired Functionality

In this section we enumerate the other potential improvements that Alliander suggested for IRMA. These are all issues that ideally should be solved in some way since they are essential to use IRMA in certain scenarios.

Credential revocation IRMA attributes have an expiry date, but for issuers a problem appears when something changes and an earlier issued attribute is not valid anymore. For example, the home address attribute changes when the user moves. In the current situation, the user can still use the attribute of his old address until the expiry date is passed. This is especially problematic for IRMA signatures, because now messages can be signed with attributes that in reality do not hold anymore. Due to this, the validity guarantees of signatures are at stake and this is an issue for Alliander.

In section 6.2.1 we described some technical solutions known from literature that can be used for credential revocation in IRMA. The options we described and the underlying overview made by Lampon *et al.* [19] can be used as a starting point to do research about what would suit best for IRMA.

Alternatives for people without a smartphone For now IRMA can only be used as an app for smartphones and tablets. This means it cannot be used by people that do not have such devices. For example, elderly people often do not know how to use a smartphone and disabled people might not be able to use an app at all. An open research topic is still whether it possible to get IRMA working on other platforms (e.g. a web application) without unacceptably giving in on confidentiality and integrity guarantees.

Using IRMA when being in conservatorship of when being mandated In some use cases, someone else should be able to manage and control the attributes of a particular person. In case of Alliander a variant on this problem occurs when attributes of a house (for example stored in a smart electricity meter) have to be managed. Solutions to this problem might require that credentials can be delegated to other IRMA users.

Bibliography

[1] Christopher Allen. The path to self-sovereign identity. Github, 2016. Retrieved from: `https://github.com/ChristopherA/self-sovereign-identity/blob/master/ThePathToSelf-SovereignIdentity.md`.

[2] Daniel J Bernstein. Cryptography in nacl. *Networking and Cryptography library*, 3:385, 2009.

[3] Privacy by Design Foundation. IRMA, n.d. Retrieved from: `https://privacybydesign.foundation/irma-explanation/`.

[4] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical report/Dept. of Computer Science, ETH Zürich*, 260, 1997.

[5] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 21–30. ACM, 2002.

[6] Kim Cameron. The laws of identity–as of 5/12/2005. *Microsoft Corporation*, 2005.

[7] CryptID. Website. Retrieved from: `http://cryptid.xyz/`.

[8] Don Davis. Compliance defects in public-key cryptography. In *Proceedings 6th Usenix Security Symposium, San Jose CA*. Usenix, 1996.

[9] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE security & privacy*, 7(1):50–57, 2009.

[10] Shayan Eskandari, Jeremy Clark, David Barrera, and Elizabeth Stobert. A first look at the usability of bitcoin key management. *arXiv preprint arXiv:1802.04351*, 2018.

[11] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.

[12] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.

[13] International Organization for Standardization and International Electrotechnical Commission. Information technology – automatic identification and data capture techniques – bar code symbology – QR code. Standard, International Organization for Standardization, Geneva, CH, 2000.

[14] Privacy By Design Foundation. IRMA issuance and disclosure protocol. Retrieved from: http://credentials.github.io/protocols/irma-protocol/.

[15] Privacy By Design Foundation. Keyshare protocol. Retrieved from: http://credentials.github.io/protocols/keyshare-protocol/.

[16] The Selfkey Foundation. Selfkey, 2017. Retrieved from: https://github.com/Cryptorating/whitepapers/blob/master/SelfKey/selfkey-whitepaper-en.pdf.

[17] Audun Jøsang, Muhammed Al Zomai, and Suriadi Suriadi. Usability and privacy in identity management architectures. In *Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68*, pages 143–152. Australian Computer Society, Inc., 2007.

[18] Tommy Koens and Stijn Meijer. Matching identity management solutions to self-sovereign identity principles. *ING Bank*, 2018.

[19] Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. Analysis of revocation strategies for anonymous idemix credentials. In *IFIP International Conference on Communications and Multimedia Security*, pages 3–17. Springer, 2011.

[20] Christian Lundkvist, Rouven Heck, Joel Torstensson, Zac Mitton, and Michael Sena. uPort, a platform for self-sovereign identity. Technical report, uPort, 2016. Draft, retrieved from: http://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf.

[21] Toru Nakanishi, Hiroki Fujii, Yuta Hira, and Nobuo Funabiki. Revocable group signature schemes with constant costs for signing and verifying. In *International Workshop on Public Key Cryptography*, pages 463–480. Springer, 2009.

[22] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. Mnemonic code for generating deterministic keys, 2013. Retrieved from: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki.

[23] Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009.

[24] Privacy By Design Foundation. irmago GitHub repository. Retrieved from: https://github.com/privacybydesign/irmago.

[25] Drummond Reed, Jason Law, and Daniel Hardman. The technical foundations of Sovrin. Technical report, Sovrin, 2016. Retrieved from: https://www.evernym.com/wp-content/uploads/2017/07/The-Technical-Foundations-of-Sovrin.pdf.

[26] European Union. Charter of fundamental rights of the european union, 2000. 2000/C 364/01.

[27] European Union. ePrivacy directive, 2002. Directive (EU) 2002/58/EC.

[28] European Union. Proposal for a regulation on privacy and electronic communications (ePrivacy regulation), 2007. EU Document 2017/0003 (COD), retrieved from: http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=41241.

[29] European Union. eIDAS regulation, 2014. Regulation (EU) 910/2014.

[30] European Union. Setting out minimum technical specifications and procedures for assurance levels for electronic identification means pursuant to article 8(3) of regulation (EU) no. 910/2014, 2015. Commission Implementing Regulation (EU) 2015/1502.

[31] European Union. General data protection regulation, 2016. Regulation (EU) 2016/679.

[32] Pim Vullers and Gergely Alpár. Efficient selective disclosure on smart cards using idemix. In *IFIP Working Conference on Policies and Research in Identity Management*, pages 53–67. Springer, 2013.

[33] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of PGP 5.0. In *USENIX Security Symposium*, volume 348, 1999.

[34] Phillip Windley. How Sovrin works. Technical report, Sovrin, 2017. Retrieved from: `https://www.evernym.com/wp-content/uploads/2017/07/How-Sovrin-Works.pdf`.

A. Detailed Scoring of Key Management Solutions

QR code on paper

Criterion	**Judgement**	**Explanation**
Full recovery	+	For encryption using a key directly stored in a QR, no other parties are involved but the user. There are also no limitations in what can be encrypted. The only limitation is that a printer is needed to print the QR.
Understandable and usable	++	QRs are a familiar concept in IRMA and printing them on paper makes it easy for users to understand how to store them safely.
Transparency	++	All techniques can be open-source and no parties have to be involved
Control	+/−	No access control can be done. If the QR on paper and a back-up are stolen, all data can be decrypted and possibly the identity can be restored without the user's permission. This however requires physical access to the location where the QR is stored.
Existence	n.a.	*Device revocation techniques deal with the existence guarantees (section 6.2)*
Privacy and data protection	−	When the QR is generated, it must be sent to a printer somehow. This transport is hard to secure, for sure for public printers.
Portability	++	Encryption and decryption is fully user-side and can be done by other apps too

Key stored on security token

Criterion	**Judgement**	**Explanation**
Full recovery	+	In the encryption process, no other parties are involved except for the user. There are no limitations in what can be encrypted. However, a token must be obtained before this solution can be used.
Understandable and Usable	+/−	This depends on what type of device and what type of security token is used. The supported techniques of both the device and the security token influence the user experience.
Transparency	+	No other parties are involved. Only limitation is that most security tokens are closed source.
Control	+	A security token can be equipped with a diversity of two factor authentication mechanisms (i.a. PIN or password protection). However, a fallback when users forget their PIN or password is hard to realize.
Existence	n.a.	*Device revocation techniques deal with the existence guarantees*
Privacy and Data Protection	+/−	When the token can be directly connected to the device with a secure connection, the key and the data can be protected. When extra devices are needed to connect the device containing the IRMA app and the security token, it becomes harder since intermediate devices have to be protected too.
Portability	+	Encryption and decryption are in control of the user, so therefore there are no dependencies on other parties. However, the choice of available security tokens might be limited. It therefore depends on the token how much choice there is.

Mnemonic phrase

Criterion	**Judgement**	**Explanation**
Full recovery	++	In this encryption process no other parties are involved except for the user. There are also no limitations in what can be encrypted. The amount of data that a phrase can contain without getting unusably long, is limited. Although, in most cases data can also be stored in the back-up file.
Understandable and Usable	+	The method is very understandable, since users can do key management on paper. Writing down the words involves some extra work of the user, so it is a bit more cumbersome qua usability.
Transparency	++	The method to convert data to words is publicly available and the keys being stored can be just keys of well-known encryption mechanisms.
Control	+/−	No access control can be done. If the mnemonic phrase and a back-up are stolen, all data can be decrypted and possibly the identity can be restored then without the user's permission.
Existence	n.a.	*Device revocation techniques deal with the existence guarantees*
Privacy and Data Protection	++	Back-ups can be immediately encrypted by the IRMA app and the key is transferred from the device to paper manually. When doing this in a private environment such that nobody can look the user over the shoulder, the data is safe.
Portability	++	Encryption and decryption is fully user-side and can be done by other apps too

Secret sharing

Criterion	**Judgement**	**Explanation**
Full recovery	++	Parties that are involved are trustees and therefore there is no limitation for what purpose the key they manage together is used.
Understandable and Usable	−	This type of key management gives extra overhead. Trustees have to be found that manage a part of your key and when recovery is needed, those trustees have to be all contacted again. In case of user trustees, these users have to be instructed how they should properly manage someone else's key without accidently giving an attacker permission to recover someone's IRMA account.
Transparency	+/−	The techniques behind secret sharing are difficult to explain. On one hand this is a limitation, but from the other side users do not have to know the technical details. They should understand what the principle is and what is expected from them. It therefore really depends on how it is implemented and if the user and his trustees are able to understand the implementation.
Control	-	Strictly, the user is not in control but his trustees are. When the trustees conspire and turn against the user, he has a problem. This is a limitation.
Existence	n.a.	*Device revocation techniques deal with the existence guarantees*
Privacy and Data Protection	+/−	Just as at control, there is a risk that the trustees might be able to conspire, optain the key and decrypt the back-up.
Portability	+	The user himself decides who he trusts. He can always decide to re-configure recovery and choose new trustees. However, especially for user trustees, re-configuration gives a barrier since new trustees should be contacted.

Trusted party manages key

Criterion	**Judgement**	**Explanation**
Full recovery	+/−	For a variety of reasons, trusted parties might be only able to recover a part of the credentials a user have. This can have a technical reason, for example keyshare servers can only participate in recovery of credentials managed by them. It can also be policy, for example certain parties might not want to intervene in credentials from other parties. However, when a trusted party is just managing a key without any further involvement no limitations exist.
Understandable and Usable	++	Trusted parties can make clear management interfaces and even give customer support to the user.
Transparency	−	User depends on the transparency of the trusted party.
Control	+/−	A trusted party can introduce extensive control features to the user with more options than all other solutions. However, this is no full control since the trusted party is in the middle and can also abuse this role.
Existence	n.a.	*Device revocation techniques deal with the existence guarantees*
Privacy and Data Protection	+/−	The trusted party can give privacy and data protection guarantees, but also here the trusted party is always in the middle and therefore there is data breach risk.
Portability	−−	The trusted party has a power relation with the user and can prevent portability. This is something which can only be realized by regulation.

B. Detailed Scoring of Device Revocation Solutions

Recovery key in credential (option I)

Criterion	Judgement				Explanation
	Option A	Option B	Option C	Option D	
Full recovery	−			−−	Recovery is only possible a limited number of times. In option D additionally the particular scheme should have set a trusted party that deals with revocation.
Understandable and Usable	−−	−	−	−	User should manage multiple secret keys. In case of option A a new key can only be used from the start of a new epoch.
Transparency	++	+	++	+/−	Current IRMA flows keeps rather the same, only a revocation list check is added. In option A and C the user is involved in this proof, so this is more visible than when the verifier does it. Then the revocation list should be made transparent to the user. In option D it depends on the transparency of the trusted party.
Control	−			+	User is in control over his keys, only individual issuers determine the revocation list for their credentials. This means some control is only transferred to issuers. In option D there is only one party to which some control is transferred: the trusted party.
Existence	+	++	++	+	In option A until C issuers have more control over the credentials they have issued because they can also revoke for other reasons. In option A it can only take some time before a change comes into effect. In option D the trusted party guarantees existence on behalf of all issuers. Credentials can not be revoked individually in option D.
Privacy and Data Protection	−−	−	−	+	All issuers have to be notified when a user starts the recovery process. In option A the user also becomes more traceable, because more communication is needed between the issuers and the user. In option D no extra communication with issuers is needed, so when the trusted party has privacy-friendly procedures this is fine.
Portability	+			−	Each issuer deals with revocation for himself. In option D there is a dependency on the trusted party.

Recovery key in credential (option II)					
Criterion	Judgement				Explanation
	Option A	Option B	Option C	Option D	
Full recovery	+/−			n.a.	Recovery is possible for a credential if the particular issuer supports it.
Understandable and Usable	+/−				Refreshment is needed. This works nice when all issuers co-operate, but in case of disruptions, even only at one issuer, problems can be experienced. In case of option A it also takes some time after refreshment before the old credentials are revoked. When the refreshed credentials are issued immediately this is not a problem, but if users have to wait this can be problematic.
Transparency	++	+	++		Refreshment is just a alternative flow of issuance, so the key concepts keep the same. In option A and C the user is involved in this proof for revocation, so this is more visible than when the verifier does it. Then the revocation list should be made transparent to the user.
Control	−				User is in control over his recovery key, only individual issuers determine the revocation list for their credentials. This means some control is transferred to issuers.
Existence	−−	−	−		Issuers have more control over the credentials they have issued. In option A it can only take some time before a change comes into effect. This means that either a user has to wait until this happens (the safe option) or for the time being two valid versions of a credential exist. However, without a shared source for information it cannot be guaranteed that the user manipulates the secret key and in that way combines credentials belonging to different IRMA accounts.
Privacy and Data Protection	−−	−	−		All issuers have to be notified when a user starts the recovery process. In option A the user also becomes more traceable, because more communication is needed between the issuers and the user.
Portability	+				Each issuer deals with revocation for himself.

Via keyshare server					
Criterion	Judgement				Explanation
	Option A	Option B	Option C	Option D	
Full recovery	n.a.	n.a.	n.a.	+	Only the credentials that use a keyshare server can be recovered. At the moment this holds for all credentials except the demo credentials.
Understandable and Usable				++	The only visible change for the user is the device revocation process. This can be done using a web service at the keyshare server, so this can be made as user-friendly as any other web service.
Transparency				+/-	Depends on the transparency of processes at the keyshare server.
Control				+	The user is dependent on the keyshare server for device revocation. However, it is already dependent on the keyshare server now, so the facto nothing changes. Also, since there is only one party it is clear to the user who he has to contact in case of issues.
Existence				+	Similar as in the current situation.
Privacy and Data Protection				+	Keyshare server can keep using the same technologies to guarantee the user's privacy for revocation as well.
Portability				-	User cannot switch from keyshare server. Only the scheme manager can do so. For scheme managers there is freedom to choose.

Credential revocation based on distributed revocation list					
Criterion	Judgement				Explanation
	Option A	Option B	Option C	Option D	
Full recovery	+/−			n.a.	Recovery is possible for a credential if the particular issuer supports it.
Understandable and Usable	+				Revocation has to be reported to the network that together manage the revocation list. Then all issuers have to be notified a certain device has been revoked. Issuers are then responsible for revoking the credentials that are linked to this device and verifiers must get the updated revocation lists from the issuers before they can notify it. The process the user has to do, can be a single web service and can therefore be user-friendly. However, users must understand that after revocation it can take a while before all issuers and verifiers are informed.
Transparency	−	−−	−		Transparency depends on transparency of the distributed revocation list. For example a distributed ledger can be transparent. This depends on which parties manage it and how the validity of the ledger is checked. After that step, issuers should also contact the distributed ledger and in case of option B users are also reliant on how verifiers interpret the revocation lists.
Control	+/−	−	+/−		The network of parties that manage the distributed ledger, the issuers and in case of option B even the verifiers get an extra role in device revocation. This is all control the user gives in. However, when problems occur the user can always complain parties do not follow the central source.
Existence	+	++	++		Issuers have more control over the credentials they have issued. In option A it can only take some time before a change comes into effect.
Privacy and Data Protection	−	+/−	+/−		All issuers still have to be notified, but this goes via the distributed ledger. When in the ledger privacy enhancing technologies are applied to guarantee the privacy of the user in the rest of the process, it can be okay. It is however unclear to what extend this can be technically enforced. In option A however communication between the issuer and the user is needed, so in here the issuer has an extra opportunity to find out.
Portability	+/−				Each issuer deals with revocation for himself, but relies on the ledger. How portable the solution is for the user depends on which party manages the ledger. When this is done in a private way, the user still cannot do anything.

Device credential issued by trusted party					
Criterion	Judgement				Explanation
	Option A	Option B	Option C	Option D	
Full recovery	++				Only credentials from schemes that issue device credentials can be recovered. When this is a requirement, probably most schemes will implement this.
Understandable and Usable	+				When recovery is started, only the issuers of the device credentials have to be informed. When these parties have clear web interfaces to do this, the usability is clear and understandable. Although, the verifiers are required to check for the device credential, so when they don't do that and proofs become accepted this can cause confusion.
Transparency	+	−	+	+/−	Transparency depends on transparency of the trusted party. In option A and C the user is involved in this proof for revocation, so this is more visible than when the verifier does it. In option D the central party does it immediately, so this is a compromise between both.
Control	+				The user is dependent on the trusted party for revocation. Therefore he gives away a little bit control. However, the trusted party does not gain all control and furthermore no parties have to be involved.
Existence	−	+/−	+/−	+	Similar as in the current situation. In solution A however it can take until the next epoch starts before revocation comes into effect. In solution B and C the verifiers have to update the revocation list.
Privacy and Data Protection	+				When the trusted party uses technologies to guarantee the user's privacy for revocation, it is okay.
Portability	−				User cannot switch from keyshare server. Only the scheme manager can do so. For scheme managers there is freedom to choose.