Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

armasuisse
Science and technology

KERI Under Scrutiny: Use-Case Study for the Swiss e-ID

Master Thesis

Lucas Falardi

September 3, 2024

Supervised by:
Dr. Martin Burkhart (CYD Campus)
Dr. Kari Kostiainen (ETH), Carolin Beer (ETH), Prof. Dr. Srdjan Capkun (ETH)

Cyber-Defence Campus, armasuisse Science and Technology
Department of Computer Science, ETH Zürich

CYD
CYBER DEFENCE CAMPUS

Abstract

This thesis presents an overview and an analysis of the distributed key management system KERI (Key Event Receipt Infrastructure) and its ecosystem with a focus on security and scalability. With the goal of providing the basis for a digital identity system for Switzerland, a KERI Network model is proposed and analyzed. The model is run on a large network of up to 10'000 users, the first time KERI has been tested at this scale. By using the keriox rust library we also contributed to its development. The final recommendation is that KERI is not suitable for the use in the Swiss e-ID base registry.

Contents

Chapter 1

Introduction

KERI [17], or Key Event Receipt Infrastructure, is a Distributed Key Management Infrastructure (DKMI) designed by Samuel M. Smith and first published in 2019. Since then, it has been presented on various conferences about digital identities[1] and people started to talk about it on the internet[2]. Despite the increasing interest, no academic research has focused on it and few resources are available to understand what it is. KERI's official specification has not yet being published, but a whole ecosystem is growing with it. The Key Management system is deployed and in use at GLEIF, an international organization responsible to release identifiers (vLEI) for companies worldwide.

One aim of this thesis is to provide a first exploitative analysis about KERI. Not only we provide a concise overview of the system, but we compare it to similar systems and discuss some limitations, in particular concerning security and scalability.

Switzerland desires to follow other nations in the introduction of a national electronic identity. After the e-ID Act of 2021 was rejected by a public vote due to data protection concerns, the Swiss government started working on a new proposal. During the consultations, the name of KERI was cited more than once as a potential component for a new e-ID infrastructure based on SSI principles[3]. KERI is described as a DKMI but most of the interest is on its extension as an identity or credentials system.

This thesis looks at KERI as a possible basic block of the future swiss e-ID system. An extensive analysis in this perspective is presented. While it is specific for Switzerland, the principles can be extended to any national-scale

[1] DICE and DICE Europe

[2] KERI jargon in a nutshell (March 2023), KERI: Key Event Receipt Infrastructure (March 2021)

[3] Technology update (8. September 2023), Meeting minutes (21. September 2023), Meeting minutes (16. October 2023),

identity system. A goal of this thesis is to evaluate the suitability of KERI for the swiss e-ID.

To complete our study we conduct a network experiment by running for the first time KERI on a large scale network. We decided to rely our implementation on the keriox library in rust and at the same time we contributed to its development. Based on the model presented in the thesis, we run KERI with up to 10'000 users over 1000 virtual machines on 8 different AWS data center around Europe.

Based on the results of our evaluation, we came to the conclusion that KERI it is not suitable as a base registry for the Swiss e-ID, due to its complexity for this use-case and limitation of consensus and privacy. In general, it is not suitable for systems with short-lived transactions and strict time requirements, as the propagation of information in the network requires some time.

Chapter 2 introduces some background information and context about the swiss e-ID and comparable systems. Chapter 3 gives a concise explanation of KERI and its ecosystem. Chapter 4 is the core part of the thesis where we present our analysis in general and in particular for the swiss e-ID. Chapter 5 shows the network experiment and its results. In chapter 6 we conclude by giving our final recommendations.

We want to mention that thesis is the result of six months of full time work including paper readings, exploration, designing, coding, debugging and multiple in-person or remote interactions with people working on KERI or on the swiss e-ID.

Chapter 2

Background

In this chapter we present some background information. To place the the thesis the context, we introduce the concept of Self Sovereign Identity and the swiss e-ID. Furthermore we explain the public key infrastructure, certificate transparency and Hyperldger Indy, since we will use them as a comparison in the next chapters.

2.1 Identity Management and Self Sovereign Identity

An identity in the digital world may come in different forms, design principles and implementations. When we make use of a digital identity we usually delegate its management to some service. We might believe we have the control over our identity, because an authentication method, such as username and password, is required to do anything related to it. As an example logging-in into a webmail service with our secret credentials and a multi-factor authentication (MFA) method gives us some confidence that no one else will be able to send emails from our account (assuming email addresses cannot be spoofed). But here in fact, we are implicitly trusting the webmail service, as in general nothing prevents the service to impersonate us and send an email from our address. This service-centric approach is illustrated in Figure 2.1.

The Single Sign-On (SSO) schema, which gives us the possibility to log-in into different services by using the same identity (or the same account), brings the benefit of reducing the number of credentials a user has to manage. On the other side, it concentrates the control of our identity in one single place and we have to trust even more the identity provider not to use our name to impersonate us on other services. We may find very convenient to use our google account to log-in on multiple websites, including our online banking account, but we must be aware that Google could always impersonate ourself and collect data about our behavior, invading our privacy.

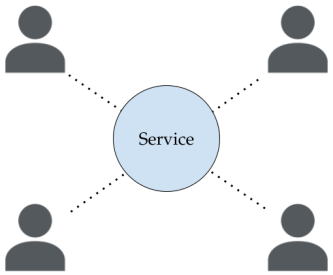

Figure 2.1: Standard approach: service-centric

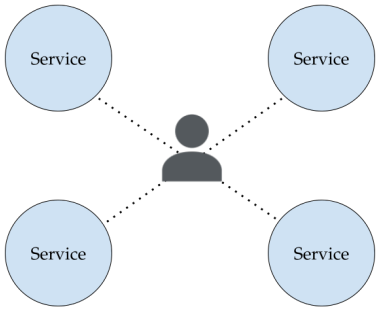

Figure 2.2: SSI: user-centric

Whether we should trust a service about our identity it is more a policy or an ethical question rather than a technical issue. One might argue that it is fine to trust Google when dealing with our YouTube account, but what if our identity is used to take decisions about our healthcare or about our bank account? This is the motivation behind the *Self Sovereign Identity* (SSI) model, which states that the user itself should be the only controller of its own identity (see figure 2.2 in comparison with figure 2.1). A complete introduction over SSI can be found in [1].

From a technical point of view, we can achieve this by using cryptography, decentralization and secured direct communications. In particular proper cryptography gives us the possibility, as long as we keep our secret key secret, to sign messages and certificates to provide integrity and authenticity.

SSI relies on the Verifiable Credentials (VC), a digital certificate, such as a passport or a diploma, usually issued by an institution and verified by some service. The simplest SSI model distinguishes three roles (see figure 2.3):

- **Holder**: the entity requesting and presenting a VC. For example a student with a Bachelor degree.
- **Issuer**: issues a VC. For example an university releasing a Bachelor degree to a student.
- **Verifier**: the entity interested in verifying a VC. For example an employer verifying a Bachelor degree of a candidate.

The roles rely on some kind of **decentralized network**, such as a public blockchain, a distributed ledger or a KERI network. This is necessary in order to reach some degree of agreement and detect malicious behaviour of the actors. The choice of this underlying network is crucial, because more centralized the network, the more the controller of the network can interfere with our identity, for example by preventing our identity from being verified.

The identities, which usually acquire meaning only when associated with a VC, are identified by decentralized identifiers (DID) [16]. Many types of DID have been proposed.

To clarify, in SSI an identity is associated to a DID which should have a strong binding with an associated cryptographic key. The controller of an identity keeps its secret key on itself, for example in a *smart wallet* in its smartphone. This key should give full control over the identity, in the sense that only the holder of the key is capable to operate with the associated identity. A decentralized network is needed to share and ideally agree on the public keys.

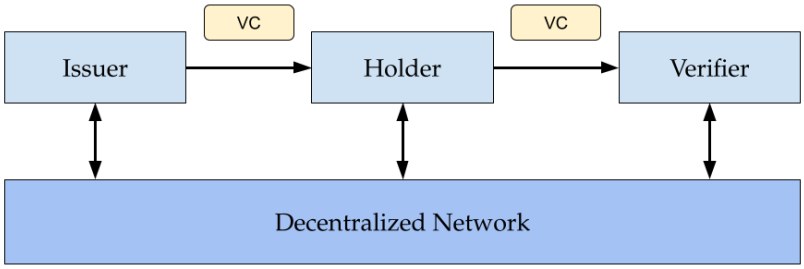

Figure 2.3: Overview of SSI roles

2.2 Swiss e-ID

The Swiss government is planning to introduce a Swiss electronic identity (e-ID) by 2026. The e-ID infrastructure should be issued and operated by the state, providing a trust infrastructure for the convenience of the state, citizens and industry. This infrastructure could then be used to build an entire ecosystem. A working group has been commissioned to define a technical solution for the implementation. The whole process has been done in an open way, including interactive feedback from the public.

Details are reported and updated on the online discussion paper [10]. Here we summarize the most important points, updated at the time of writing this thesis. The swiss e-ID fixes the following design principles: privacy by design, data minimisation and decentralised data storage. International compatibility is considered as well.

Two scenarios are possible. In scenario A the swiss system would follow the technology indicated by the EU's solutions in the field of electronic identity. However the public has expressed uncertainty on the privacy proprieties of such direction. For this reason a second scenario, scenario B is proposed.

In the latter an higher level of privacy would be offered to holders, with a particular focus on unlinkability.

The infrastructure high level design follows the SSI roles:

- The **issuer** is composed of a federal identity issuing service and a federal issuing framework.
- The **holder** is made of a wallet and a back-up system
- The **verifier** is a Check-App or an authentication service AGOV (public service login for Swirterland)
- The **registry** as the combination of the base-registry (the **network**) and the trust-registry

The swiss e-ID law draft [3] describes the required functionalities of the base and the trust registry.

- The **base registry** should provide the necessary data, such as cryptographic keys or DIDs, to issuers and verifiers, such that it is possible to verify the authenticity and status of a credential presentation. This includes a privacy preserving revocation method. Note that this registry only provides origin authentication of the credentials, in other words it ensures that a certain credential has not been revoked, has been issued from some identifier A and has been presented from some identifier B. It does not tell anything about the veracity of the content of the credential, nor about the authority of the issuer A.
- The **trust registry** should provide the necessary data to verify the identity of an issuer or a verifier. In other words it certifies that a specific DID is controlled by some specific entity, such as a company, a government office or a physical person.

Furthermore, the discussion paper reports technologies such as OpenID4VC [24], SD-JWT [7], JSON-LD [4] and BBS+ Signatures [13]. They report negative experiences as well, such as with the *Hyperledger Indy Stack*, which in two pilot projects showed performance and scalability issues, despite the good privacy proprieties.

KERI and ACDC credentials are mentioned as "other candidates", and in particular it is written that "*the concepts of KERI might be considered while conceiving the base registry*".

2.3 Public Key Infrastructure

Before introducing Certificate Transparency (CT), we write a quick introduction on the Web Public Key Infrastructure (PKI).

The Web Public Key Infrastructure (PKI), together with Certificate Authorities (CA) and X.509 Certificates, is the backbone of today's internet security. The goal of the PKI is to ensure that a client request of a web content from a specific domain, such as *example.com*, is actually coming from the controller of that domain. This is particularly important to prevent man-in-the-middle or hijacking attacks.

This model follows a hierarchical structure with multi roots. The roots are trusted CA which are usually pre-installed in browsers or operating systems in the form of root certificates. A X.509 Certificate binds a domain to some public key. The certificate is valid if it is issued by a trusted CA. The CA must verify the authenticity of the domain's owner before issuing the certificate. This can be done in many ways, such as a phone call to the registered owner or an automated verification method such as ACME [2].

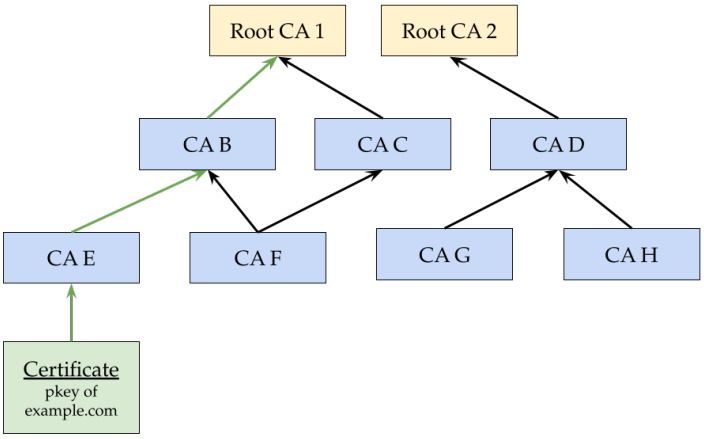

Figure 2.4: An example of PKI hierarchy. The green arrows show the chain of trust of the certificate.

A trusted CA is a CA owning a trusted intermediate certificate. A trusted intermediate certificate can be issued by another trusted CA or a root CA. A certificate is only valid when there is a direct path of trusted and valid certificates up to a root. This is called *chain of trust*.

2.3.1 Certificate Transparency

It is clear that the PKI hierarchical model exposes some weaknesses, in particular regarding trust and security. Any CA with a valid intermediate certificate is capable to issue a valid certificate which is accepted by most of the devices in the world. What if this CA is compromised, what if there is a bug in their management system? Those are not only remote possibilities, but concrete attacks which happened in the past. One example among

others is the famous DigiNotar Attack of 2011 [14]. An attacker was able to compromise the dutch DigiNotar CA and issue numerous fake certificates, used for man-in-the-middle attacks.

We can affirm that the web PKI's trust hierarchy lacks of auditing on the issued certificates. Based on this necessity, CT was introduced [12]. The core idea of CT is to make every certificate issuance transparent, such that anyone can observe it, making the domain owner eventually aware of rogue certificates.

We show how CT works with an example.

1. The domain owner requests a certificate to some CA.
2. The CA runs some verification process and then releases a pre-certificate to the logs. The logs are publicly accessible and append only. They release a Signed Certificate Timestamp, a promise to insert the provided pre-certificate into the log within the Maximum Merge Delay (MMD), usually 24 hours.
3. Finally the CA issues a certificate and provides the collected SCTs to the domain owner.
4. Every time a user visit the domain, the owner presents the certificate along with SCTs. Ideally, the user only accepts the certificate if a sufficient number of SCTs from trusted logs are presented. The trusted logs are usually hardcoded in the browser or the OS, in the same manner as for the root CAs.
5. Meanwhile the so called Monitors, independent entities, periodically check the logs (5) looking for new issued certificates and notify the domain owner when they observe new ones.

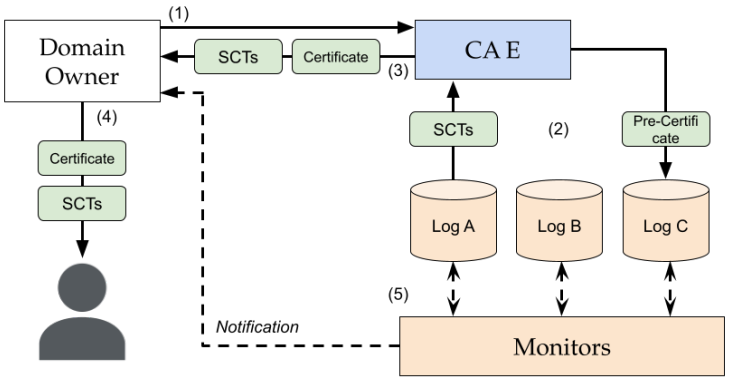

Figure 2.5: Overview on CT.

Intuitively CT works because a rogue certificate has to be inserted into the

logs, otherwise no valid SCTs can be presented, and monitors will eventually detect it and notify the domain owner, who will take countermeasures. CT is not able to directly stop an attack, but it makes it evident. Nowadays it is implemented in most of the modern web browsers. However the check on the SCTs is not always enforced, meaning that a domain providing only a certificate could still be accepted. A study from 2019 gives an overview on CT adoption in the web [23].

KERI as presented and discussed later in this thesis has a lot of similarities with CT.

2.4 Hyperledger Indy

Hyperledger Indy [11] is a distributed ledger for digital decentralized identities. It is part of the Hyperledger framework[1], a set of projects in the field of blockchain implementations. It is meant as an industry solution, providing code and tools for the design of identity systems. Correlated projects are Hyperledger AnonCreds, a format of Verifiable Credentials with privacy in mind and Hyperledger Aries, a toolkit for managing distributed identities and credentials, including code for agent applications (issuers, holders and verifies).

The Hyperledger Indy ledger follows a PBFT (Practical Byzantine Fault Tolerance)-like consensus protocol. On a very high level, its architecture is composed by two pools:

- **Validator Pool**: set of machines handling writes and reads on the ledger. They run a Redundant Byzantine Fault Tolerance (RBFT) protocol, an improved PBFT with multiple instances executing in parallel.
- **Observer Pool**: set of machines handling the reads from the ledger. They are syncronized with the validator pool. They are meant to scale reads from the ledger.

A client should send write requests to the validator pool and read requests to the obeserver pool. Furthermore Indy has different ledger types for different purposes. Each ledger is made of a transaction log and a merkle tree. Authentication and authorization are based on the information present on the ledgers.

Hyperledger Indy presents itself as a good solution for SSI, delivering some code ready to be used. On the otherside the fact that its concept is based on a PBFT consensus exposes it to the natural issues of distributed consensus regarding scalability. An experiment conducted in [5] shows the scalability

[1] https://www.hyperledger.org/

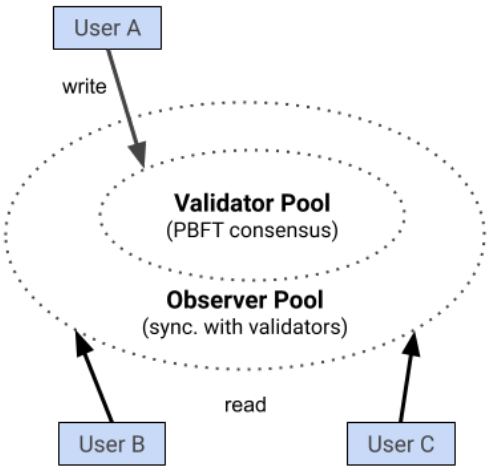

Figure 2.6: Hyperledger Indy core architecture

limitations of Hyperledger Indy, suggesting it is not suitable for a large identity system, such as a state-wide (swiss e-ID) one.

Chapter 3

KERI Overview

KERI's main goal is to provide a way to establish the current public key of an identifier and give to the identifiers owner full control and autonomy over it. This is archived with the help of a distributed and decentralized network, in which different roles coordinate to share cryptographic keys and protect the users from threats. With a few extensions the same infrastructure can be used to implement an SSI system claiming better scalability, better self-sovereignty and additional security properties. This solution offers an alternative to standard blockchain ledger-based SSI approaches, which have known limitations in scalability. For this reason we are interested in KERI as an approach for the swiss e-ID.

In this section we introduce KERI and its ecosystem in a compact and clear way. We also want to understand if and how KERI is already used in practice. This section is the result of an initial exploration phase, in where we analyzed papers, online resources and had meetings with some of the people working on KERI.

3.1 KERI Concepts

Key Event Receipt Infrastructure (KERI) is primarily a Decentralized Key Management Infrastructure (DKMI), first presented by Samuel Smith in [17]. The white paper is rather long to read and goes into many details on different abstraction levels. In the following we summarize the main ideas of the high level concept of KERI.

3.1.1 KERI Identity

A KERI identifier is called an Autonomic Identifier (AID), a form of Decentralized Identifier (DID) and it is a digest of some initial public key. In contrast to most of the classical approaches where the identifier is bounded to the same

initial keypair forever, KERI associate an AID with a chained list of events, which are capable of changing the associated current authoritative key. This list is called a *Key Event Log (KEL)*, and can be seen as a personal blockchain of an identifier, since its content it is cryptographically chained and it is owned by a single identifier. We call the KERI binding of an identifier with a KEL a KERI identity.

A user can create an identity by forging a new key-pair, the relative self-certified AID and a so called *Inception Event*, the first entry of the identifier's KEL. The latter specifies the initial public key, as well as other initial settings. We call a controller the owner of an identity, or the owner of the identity's public key.

The controller of an identity can change the current public key associated with its identifier by adding a *Rotation Event* to its KEL. This can be done in a secure way thanks to a pre-committed key authenticating the controller and makes the identity mutable over time. This feature allows the controller to recover from a compromise of the current key, to upgrade to a more secure key or just rotate the current key.

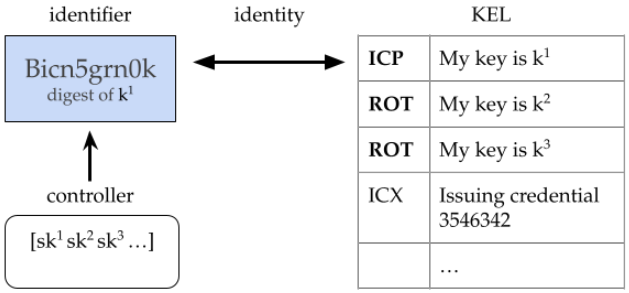

Figure 3.1: The binding of a KERI identifier to a KEL constitutes a KERI identity. The controller owns the secret keys.

Furthermore, a controller can add any other type of data to the identifier's KEL, by adding a so called *Interaction Event*. This allows for many extensions of the system. An example is using an Interaction Event to share the issuance or the revocation of a verifiable credential.

A controller of an identifer could directly provide its own KEL to a verifier, an entity who wants to establish the identifier's current key. This would require the controller to be always online and available, which is not always desirable and not always possible. Furthermore, a compromised controller could create multiple forks of a KEL, and present a different fork to a different verifiers, which could result in a security issue. For example some verifiers may see a credential as valid while others not. When a fork of the same KEL exists this is called **duplicity**.

Since the KEL is a chained list, by principle only the last few events (if not even only the last) can be kept in memory by an entity, for example a verifier, and it would be still possible to continue verifying the chain. This allows for scalability even with large KELs. In particular intermediate entities could keep a local cache of the KEL, with only the last few events. If the whole KEL or more events are required, it can be provided directly by the controller.

3.1.2 Witnesses

A witness in KERI is a role which has the purpose of witnessing a particular version of a KEL. When a witness sees a KEL for the first time, it will sign this observation by making a *Receipt*. It is important to notice that the witness, as every other entity in KERI, follows the principle of "first seen". That is, only the first seen version of a KEL is accepted, if later a fork of the same identifier's KEL is observed it is just ignored.

With multiple witnesses multiple receipts can be produced. An update to the KEL is considered to be valid only if a sufficient number of witness receipts are presented. Each controller defines the witnesses to trust in the Inception Event. A controller trusts that a witness in the sense that it would not hide your updates (entries in the KEL) and will share its receipts when possible. Furthermore the Inception Event specifies how many Receipts are required to validate an event, this value is called *tally value*. An Event is considered valid from the point of view of the controller only if at least *tally value* receipts are presented along with it. For example, out of 5 witnesses, 3 receipts might be sufficient. Since trust can change over time, the controller can change the witnesses or the tally value within a rotation event.

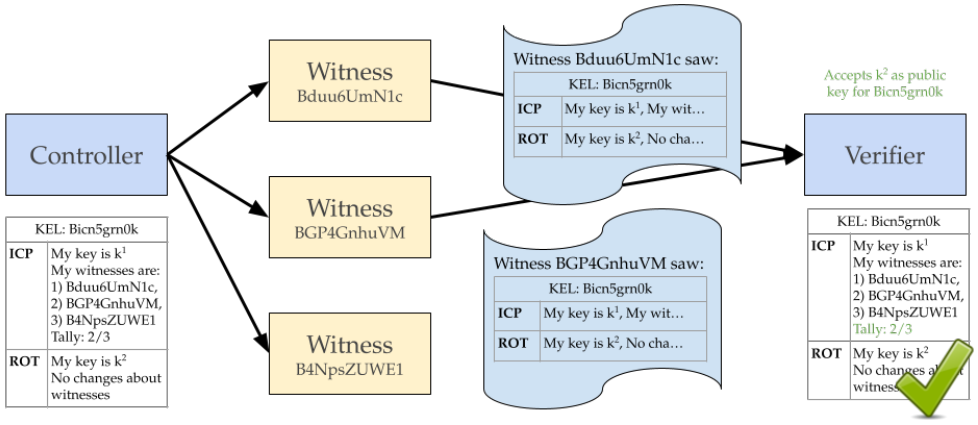

Figure 3.2: The controller sends its KEL and KEL updates to the witnesses. The witnesses provide receipts to the verifier. The verifier accepts the current KEL because it has a sufficient amount of valid receipts.

Witnesses provide a first layer of protection from duplicity, because no fork will be able to receive a sufficient amount of receipts, when a proper tally value is set. As an example, imagine an identifier x is authoritative for the issuance of some credential. x's current authoritative key has been compromised and an attacker creates a fork, let us call it KEL_2, of x's KEL (KEL_1), by adding an interaction event to revoke some credential. So, KEL_2 only has one event more than KEL_1 and is able to get receipts to validate it, since it is the first time the witnesses see this version. But when x's legit controller adds another event to KEL_1 (KEL_3), and tries to get receipts, it does not work, because witnesses already released receipts for the same version (KEL_2). Even if the attacker was able to validate its fork, x's controller is now aware that someone compromised its key. The controller can therefore rotate the current key and stop the attack. Furthermore, no inconsistency has been generated between the verifiers, since all of them agree on the (malicious) revocation, and it is not the case that some verifier would accept KEL_1 and some KEL_2.

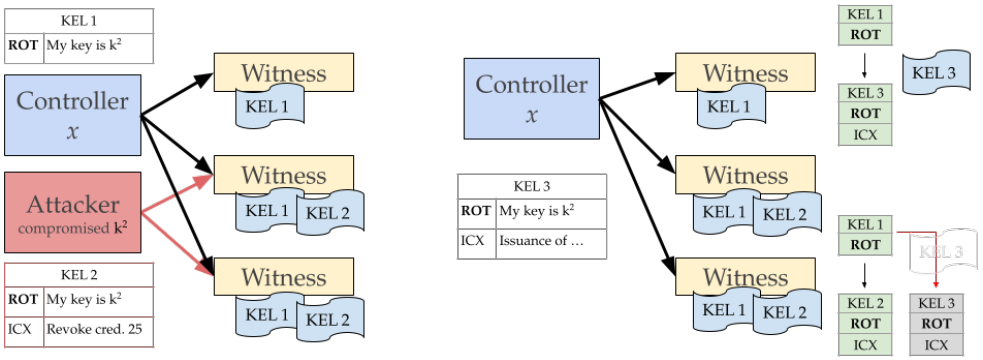

Figure 3.3: On the left: the controller first sends KEL_1 to all the three witnesses, which all produce a receipt. Then the attacker sends KEL_2 to only two witnesses, which produce a receipt, since the chain is valid (first seen). On the right: later the controller adds an event to KEL_1, we call the updated KEL KEL_3. One witness is able to validate the event and releases a receipt, while the other two see invalid sequences and cannot release a receipt. Assuming a tally value of 2/3, only KEL_2 is can be validated, so there is no inconsitency for verifiers or watchers querying the witnesses, which can only accept KEL_2 and not KEL_3.

Note that the witnesses mainly protect the controller from duplicity, as they are trusted by the controller. They even might collude with the controller.

A second purpose of witnesses is availability. A controller might not be always online, for this reason witnesses can still provide to any verifier an identifier's KEL. We use multiple witnesses to increase availability, but as well to increase security, as depending on thresholds we might tolerate that some of them are compromised or unreliable.

3.1.3 Pre-rotation and security features

A key rotation is the change of the current authoritative key associated to an identifier (AID) and can be done by adding a Rotation Event to the identifier's KEL. There can be many reasons to rotate the key, such as security over time, change of technology or compromise of the current key. But how do we guarantee that only the proper controller of an identity is able to rotate the key? KERI presents this feature called pre-rotation. When a controller makes a rotation event, or an inception event, it pre-commits a digest of the next key to be used. In practise, every time we rotate or incept an identifier, we use two key-pairs: the new current key, i.e. the pre-commited one (or new, in the case of inception) and the next key-pair, that we pre-commit by including a digest of it in the rotation event.

The pre-rotation allows the controller to regain control over its identifier even when the current key is compromised. The current key is sufficient to do any operations outside KERI, or to add interaction events to the KEL. In order to rotate, it is necessary to know the pre-committed key, which we assume is protected with higher security than the current key. In fact, since only a digest of the key is exposed, the pre-rotated key could be generated in a secure hardware device and never leave it before it is needed. The pre-rotation digest is considered to be even quantum secure.

Event	KEL: Bicn5grn0k		
0	**ICP**	My key is k^1 My witnesses Tally: 2/3 Next key: h(k^2) Chain: -	*Signed with k^1*
1	**ROT**	My key is k^2 My witnesses Tally: 2/3 Next key: h(k^3) Chain: h(Event(0))	*Signed with k^2*
2	**ROT**	My key is k^3 My witnesses Tally: 2/3 Next key: h(k^4) Chain: h(Event(1))	*Signed with k^3*

Figure 3.4: Here is a more complete example of a KEL. Backward chaining is done by including an hash of the previous event ("Chain"). Forward chaining is made with pre-rotating the next key ("Next key"). Messages are all signed with the new current key. For example, event 2 is valid because the key advertised matches the signature and the hash of the new key matches the pre-rotated key from the previous event (event 1).

KERI also provides other security features. We mention multi signature schemes, where multiple signatures from different identifiers are required to

add an event of some identifier's KEL. Delegation allows another identity to sign in behalf of another one. With delegation we can add an additional layer which enables the controller to revoke the delegation in case of compromise. Both multi signatures (KERI supplies as well weighted multi signatures) and delegation allow to use KERI to model complex hierarchy. As an example, a company requiring signatures from multiple members of the direction can be modelled, or delegating even temporarily the signature to a substitute it is possible.

KERI claims to be in general very flexible and crypto-agile. Even after an identity is created, it is possible to change the type of key in use, to better respond to technology upgrades over time.

3.1.4 Watchers

Witnesses should protect the controller from duplicity. However, a compromised controller together with compromised witnesses could still produce duplicity, in the sense that they could distribute forks of the same KEL, with valid receipts for each one. Verifiers need to be protected as well, and this job is assigned to watchers.

Watchers do the exact same job of witnesses. They observe witnesses receipts and KEL updates, but only accept the first version they see. A verifier, instead of querying the defined witnesses, relies on a set of trusted watchers. When two verifiers relies on the same watcher, or on the same set of watchers, there cannot be duplicitous KELs, as a fork of a KEL would not be accepted by the watchers in common. The verifier do not specify the watchers he uses and can every time query a different one.

Similarly as for witnesses, we want multiple watchers to operate for availability. At the same time a verifier could require a minimum number of watchers to be queried, analogously to the controller's tally value, to confirm agreement in the set of trusted watchers. Watchers can produce watcher receipts, if necessary.

3.1.5 KA2CE

The KERI's Agreement algorithm For Consensus control establishment (KA2CE) is an algorithm[1] which applies the principles of byzantine fault tolerant majority consensus to reach an agreement in a set of witnesses. An agreement here is defined as a KEL event together with sufficient receipts for the event to be validated, according to the values set by the identifiers controller. By fixing a number of possibly faulty witnesses F, and the number of witnesses trusted by the controller N, KA2CE defines the correct tally

[1] Even if it is called an "algorithm" it is more a set of rules

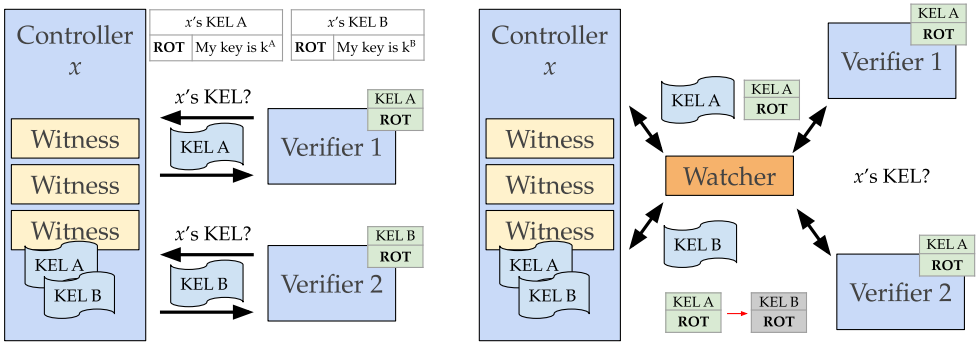

Figure 3.5: On the left a compromised controller and witnesses are able to produce duplicity resulting in Verifier 1 and 2 accepting two conflicting KELs. On the right the queries are routed through a watcher. After the query for Verifier 1, the watcher validates KEL A. If the compromised controller tries to provide KEL B to the watcher, upon request of Verifier 2, this does not work, as the watcher detects and ignore the conflicting KEL. Both Verifier 1 and Verifier 2 agrees on the KEL A.

number M to reach fault tolerant majority consensus among witnesses. It is important to mention that KA2CE does not guarantee that all witnesses agree on the same version of the KEL, but only a majority, which is necessary to create a consensus among verifiers.

Intuitively, if a controller trusts 5 witnesses but sets the tally value to 2, it is possible that 2 witnesses accept one version of the KEL, and 2 other witnessess accept another version of the KEL, since there is no consensus running between witnesses by default. However when witnesses communicate to each other, for example by disseminating receipts when releasing them, such cases can be detected. Remember that witnesses in general may be run by independent and uncorrelated entities from all around the globe. The algorithm recommends the following lower and upper bound for M:

$$\frac{N+F+1}{2} \leq M \leq N-F$$

Note that there is no way to enforce a controller to set a proper tally value. However, verifiers may decide not to accept a KEL which tally value is not compliant with the recommendation.

Old tally

In addition to this, the algorithm specifies an optional feature, which requires a number of witnesses not to change during a rotation, or in other words, after changing the witnesses with a rotation event, the rotation must be still witnessed by a certain amount of old witnesses. This is meant to increase stability of changes by not allowing too many changes at once,

which may lead the identifier to instability, if for example the new witnesses are unreachable.

Recovery

Finally the algorithm clearly states the recovery policy in the case a fork of the current KEL is detected. A fork is only recoverable when only interaction events were added. Interaction events can be added with the current authoritative key, while rotation events require the pre-rotated key to be used as well.

In particular it is specified that rotation events overrides an interaction event, and so the current authoritative KEL fork can be reestablished. Please note that interaction events which were created and witnessed in the meanwhile, before a recovery trough a rotation event, are still accountable even after recovery, meaning that a verifier still consider them as valid (remember that malicious interaction events are caused by a compromise of the identifier's key).

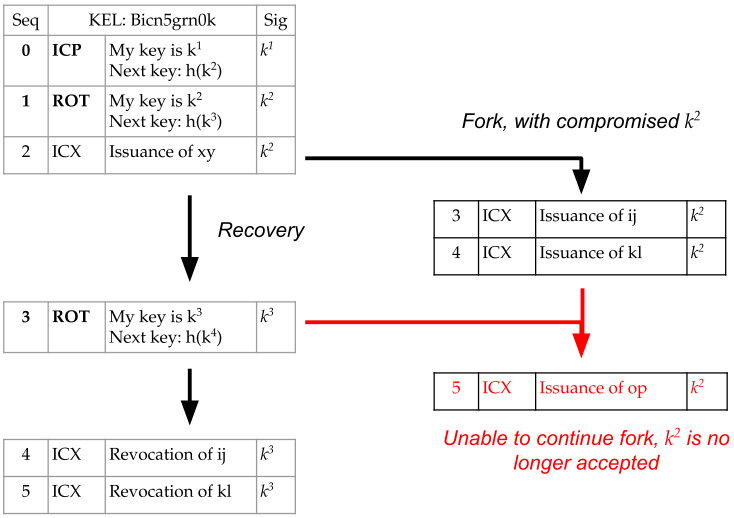

Figure 3.6: In this example, after compromise of k^2 an attacker forges the proper KEL and is able to add two interaction events (issuance of credentials ij, kl). The original controller can recover control with a rotation, which oversight in sequence interaction events and move back the chain to the legit one. However, events 3 and 4 are still accountable. This means that the controller might have to take measures, like revoking ij and kl.

3.1.6 Jurors and Judges

KERI also introduces other two roles in the network, which we quickly explain. A Juror is an entity with the object of detecting duplicity. While witnesses and watchers have the job to stop duplicity, jurors should spot it.

A juror observes the network, and in particular it observes witnesses and watchers local KELs (of other identifiers). It compares the KELs looking for duplicity, and if it is the case, keeps the proofs in a Duplicitous Event Log (DEL).

A Judge is above jurors and collects information from their DELs. A Judge may be part of the trust basis of some verifier, to ensure no duplicity is present.

3.1.7 The TEL and the SSI model

Finally we describe how to use KERI to implement an SSI model with Issuers, Holders and Verifiers. An issuer is first an identifier in KERI with its own KERI identity. In addition to the KEL, an issuer uses a Transaction Event Log (TEL), which may not be operated by the issuer itself. An issuer may issue any verifiable credential (VC) format just by using its signature, verifiable with the key established in KERI. The VC format ACDC comes along with KERI, but it is not KERI-dependent.

The TEL is a ledger used to keep track of issued or revoked credentials. It is needed since the KERI signatures may not be verifiable anymore over time, because of rotations. In particular witnesses or watchers may not keep the whole KEL history. Even if may be possible to fetch the whole KEL from the controller, it may be inefficient. For this reason we anchor any change of the TEL to the KEL, by writing into the KEL a digest of the current TEL (with many degrees of granularity). It can be done by inserting interaction events, or even by adding payloads to rotation events. In addition the TEL acts as a revocation check mechanism.

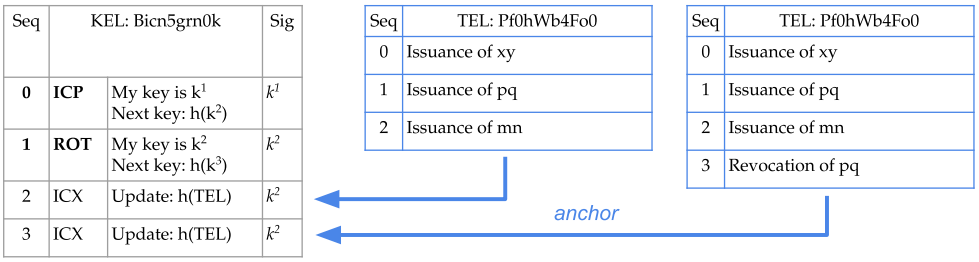

Figure 3.7: After an update in the TEL, this is anchored into the KEL by putting an hash in an interaction event.

The TEL could be implemented in many efficient ways, such as a Merkle tree, which could efficently provide the root hash to be anchored in the KEL. The fact that the TEL could reside on an external entity, might enhance privacy in certain situations. An external host of the TEL might provide higher privacy

guarantees, while the issuer would only see who is verifying but not what by getting KEL queries only. With the help of delegation we can deal with very large TELs, by stopping using one identifier and delegating to another one with its own TEL. Alternatively, multiple TELs can be anchored to the same KEL.

An holder is another KERI identity, receiving a credential from an issuer. The credential can be presented to a verifier with the IPEX protocol (a presentation protocol specific to KERI [22]).

A verifier has its own KERI identity as well, because all kind of queries in KERI have to be signed with its own key. Upon the authenticated presentation of a credential, it can verify the authenticity and the status of the credentials with a lookup to the indicated TEL and a query to confirm the TEL's authority in the issuer's KEL. Of course, the authority of an issuer must be established by another mean.

3.2 KERI Stack

KERI comes along with some credential format, encoding format and libraries, which together are supposed to extend KERI. Here is a quick overview of the work in progress related to the "KERI Stack".

ACDC

Authentic Chained Data Containers [18] are a type of Verifiable Credential. It makes use of Self-Addressing Identifiers (SAID) [21], a type of content-addressable identifiers, to recursively digest sections of data. The outermost SAID can be inserted into the TEL or the KEL of the issuer, to further authenticate the issued credential. The compartmentalized approach enables graduated discosure and ricardian contracts [9], in order to build a chain-link confidentiality, a chain of contractual restrictions between entities in an hierarchy. It is compact by design.

CESR

The Composable Event Stream Representation [19] is a binary encoding format with the unique property of text-binary concatenation composability. We have not analyzed this encoding in details, but it claims to be designed with cryptography in mind. KERI can run on any encoding, including CESR, which was designed to make KERI more efficient and minimize the amount of data in transit.

OOBI and IPEX

The Out-Of-Band-Introduction [20] and the Issuance and Presentation EXchange [22] are two important protocols used in KERI. The first is a discovery mechanism used to gather the information necessary to verify an identifier. It provides a method to get over the untrusted web, with an URL, the bootstrap data, such as the set of witnesses, of an associated KERI identifier. The second is a presentation protocol fundamental for the presentation and issuance of ACDC credentials. It allows contractually protected disclosure of the data contained in the ACDC.

We could not find a security nor a formal analysis about the correctness or the security analysis of these protocols. Such an analysis would be interesting, but it is out of the scope of this thesis.

3.2.1 KERI existing implementations

As the best of our knowledge, two KERI core libraries are in development. One offers an implementation in python, the other one a rust one. Nevertheless, not all functionalities, and in particular not all roles defined in KERI have been implemented into code.

KERIpy

KERIpy[2] is the main existing KERI implementation. Written in python, open source, it provides core functionalities of KERI. It is maintained by a group of developer connected with the Trust Over IP foundation[3]. At the moment of writing this thesis, they have implemented controllers, witnesses, ACDC, TEL, KEL, and other KERI features like multi signature schemes, delegation, and some KERI-related protocol like OOBI and IPEX. Interestingly they have no implementation for watchers, nor for jurors, nor for judges.

KERI-A

KERIA[4] is build on top of KERIpy and it is meant as a cloud solution for KERI, providing functionalities for the use of KERI by any user. "KERI Agent in the cloud" is a solution to execute the KERI protocol in the cloud, while keeping the secret key on the edge device, for example in a digital wallet on a user's smartphone. This could be a product solution for many applications, including for an e-ID.

[2] https://github.com/WebOfTrust/keripy
[3] https://trustoverip.org/
[4] https://github.com/WebOfTrust/keria

KERIox

KERIox[5] is a KERI implementation in rust. It is maintained by the human colossus foundation[6]. The implementation includes the roles of the controller, witness, verifier and watcher. Main functionalities, including ACDC, TEL, KEL, multi signature schemes, delegation are also implemented in rust. Bindings for JavaScript/Typescript and Dart are provided as well.

3.3 KERI deployment

There is one known use-case where KERI is deployed, and much interested on where could be deployed in production.

GLEIF and vLEI

The Global Legal Entity Identifier Foundation (GLEIF)[7] is an international, independent company which has the primary purpose of issuing Legal Entity Identifiers (LEI) for companies all around the globe. The LEI is a unique and standardized identification, to be used in business processes, such as for financial or digital exchanges.

The vLEI is the verifiable, digital version of a LEI. GLEIF decided to implement vLEIs based on the SSI principles. Without explaining the whole hierarchy of vLEI and vLEI issuers, we can highlight that part of the infrastructure is running using KERI [8].

Unfortunately no reports on the use of KERI are available to the public. We believe they make use of the KERIpy library. We know that their model is a KERI model based on a set of trusted witnesses, and they are running no watchers at the moment. We analyze this model later in the thesis.

EBA

The European Banking Authority[8] expressed its own interest in the use of KERI along with vLEI credentials. In this [6] discussion paper about the Pillar 3 Data Hub, they report the results of an evaluation committed to Garner Consulting. They evaluate KERI as a portable Decentralized KEy Management Infrastructure (DKMI), with innovative security features.

[5] https://github.com/THCLab/keriox
[6] https://humancolossus.foundation/
[7] https://www.gleif.org/
[8] https://www.eba.europa.eu/

Chapter 4

Analysis: KERI as foundation for Swiss e-ID

This is the core section of this thesis, where we present the analysis done on KERI. We analyze the principles of governance in KERI, we make a comparison with similar systems, and we highlight the weaknesses we identified in KERI. Then, we show the model we designed for the use in swiss e-ID, followed by an analysis on the security properties and finally a check on the e-ID requirements. We discuss design choices in details and leave some open question about KERI.

4.1 KERI and Governance

The flexibility of a KERI network allows a multitude of different designs. Beyond the technical aspects, we would like to first discuss some principles about governance, which we believe being a starting point of any design of this type. Anarchy is the total absence of governance, where no government nor laws exists. In this scenario individuals can interact peer-to-peer, but since no coordination exists, it is impossible to scale up communication and productivity. Clearly, we do not desire this property in a KERI network, or in a distributed system in general. As an example in the context of a swiss e-ID, all swiss citizens has to agree on a credential format, on a set of trusted issuers and on the infrastructure to be used.

While it is clear that some sort of governance is required, we also want it to be distributed. A centralized governance has certainly many advantages, for example it is technologically simpler to implement, but also it requires a stronger, harder to reach, trust. From a security point of view, centralization means a single point of failure, opening the door for attacks and reducing reliability. Further discussion of this topic is beyond the scope of this thesis. However, an interesting discussion on governance can be found in [15].

The technical options given by KERI, such as delegation, choice of witnesses, choice of watchers and multi signature schemes, allow implementing policies defined by the government model. Here is an example on how we could use these features to model a swiss governance.

4.1.1 A federated KERI network

Switzerland is a federal state. Compared to other nations, the power is well distributed. In principle, three authorities are recognized, each having its own jurisdiction, law enforcement and form of justice:

- Federal authority: the highest authority in the state, including the federal council and the parliament.
- Cantonal authority: cantons are large regions on the territory. They are quite independent from each other. For example, school system can differ from one canton to another.
- Communal authority: each town or village has its own independence.

Some operations require the involvement of multiple authority. For example, the issuance of a passport requires both the federal and the cantonal authority.

Based on these information, we initially came up with the following KERI network draft which tries to model the swiss governance for the swiss e-ID implementation. Here we implement the e-ID as an ACDC credential. The main components are:

- **Cantonal e-ID Issuer**: the canton would be responsible for issuing the e-ID. Multiple issuers can exist, which delegate its authority to multiple sub-issuers for availability, scalability and better post compromise recovery (if one sub-issuer is compromised, we just revoke its delegation). The number of issuers and sub-issuers can variate depending on the canton's necessities. Each sub-issuer has its own KEL and TEL. The sub-issuer's identifier partecipate in a multi-signature scheme with the so called *signers*. A scheme where a signer from the canton and a signer from the federal state, each having 50% of power in the scheme, would perfectly model the involvement of both the authorities.
- **Signer**: following the same principle as for the *issuers* with delegation and distribution, a signer could be operated by a federal or cantonal authority. It participate in a multi-signature schema for the issuance of e-ID. Its grade of power may depend on the importance of the issued credential.

We did not involve communal authorities for the sake of simplicity in this example. But expanding the model beyond swiss e-ID, many possibilities emerge. Imagine a communal authority represented by a persisted KERI

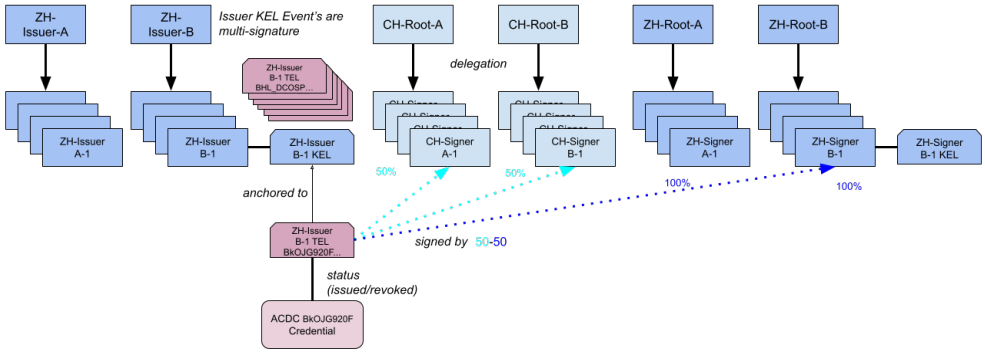

Figure 4.1: Federated KERI Network draft for the swiss e-ID. "CH" is the federal government, while "ZH" is the canton of Zurich. In this example, to validate an ACDC credential issued by the canton of Zurich, a KEL entry (anchored to a TEL entry, for illustration purposes the arrows start from the TEL entry) must be signed be one cantonal authority and one federal authority. In this example, one signature from a cantonal signer and two signatures from two federal signers would be required.

identifier. This authority could be delegated to the town's principal KERI identifier for the necessary time. Or, as another example, imagine a KERI identifier representing the parliament, where an entry in the KEL would only be valid if the majority of a multi-signature schema is reached, each signature coming from the KERI identifier of parliamentarians.

KERI is very customizable in this sense, and could be used to model various governances. For this reason we started modelling the e-ID KERI network in this direction. However, after a meeting with the swiss e-ID team, we realized that this would not meet the requirements. In particular, the swiss government it is interested in a more centralized and focused on swiss e-ID way. This decision it is understandable, considering swiss is a relative small country, and involving all the authorities could require a lot of effort for minimal benefit. Nevertheless, using KERI for a federated model is promising and might make more sense for larger nations or organizations.

4.2 Comparing KERI

To better understand the features of KERI, we compared it with two similar systems, Certificate Transparency and Hyperledger Indy.

4.2.1 KERI versus Certificate Transparency

Sam Smith, the inventor of KERI, during DICE[1] 2024 described KERI as "Certificate Transparency + decentralized Certificate Authority". This statement

[1]Digital Identity unConference Europe, 2024 Zurich

is actually a good summary about what KERI is and what is not. In the following we analize to which extend this is true.

The analogous of KERI Witnessess are CT Logs. KERI Witnesses releases receipts, CT Logs releases SCTs. In KERI an Event is accepted if a sufficient number of receipts are presented, in CT a certificate is accepted if a sufficient number of SCTs are presented. If you think about it, a log and a witness have the exact same role, witnessing that something happened. In the case of CT they witness a certificate issuance, in the case of KERI they witness a Key Event, which can be anchored to the issuance of a credential.

In CT Monitors observe the network and detect fraud, in KERI this role is taken by Watchers (and eventually Jurors, Judges).

A difference is that in CT there is a well defined set of logs, which is trusted and publicly known. All the information goes on these decentralized logs, which are operated by a few companies and are considered trusted party in the system. Note how this approach is similar to a distributed ledger, but instead of having one single ledger, we have multiple of them, each operated by a different entity.

The fact of having a limited set of trusted logs makes the job of the monitors much easier. The first reason is that you can assume the logs are not trying to fool you. In KERI, a witness could collude with the controller of some KEL, making some attack easier. The second reason is that having a limited set of logs to check is more efficient and scalable, compared to having one log, possibly each one on different Witnesses, for each identifier in the world.

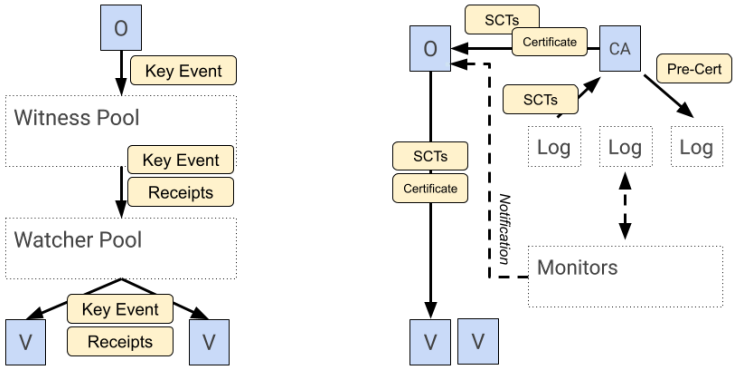

Figure 4.2: KERI (left) and CT (right). O is an owner, V a verifier.

KERI claims to be fully distributed, meaning there is effectively no central authority. This is because each identifier can determinate its own witnesses and each verifier its own watchers. However we believe that it is hard to run such a model in practise. In reality we should instantiate KERI with some degree of distribution or centralization, like in CT.

Another difference which worth mentioning is responsibilities. Remember that in CT attacks are not prevented, but only made evident to the domain owner, which has the responsibility to take countermeasures. On the other side in general in KERI much responsibility is given to the verifier. This is not always desired, such as in a industry system, as it could be the swiss e-ID, where institutions like banks or healthcare structures would join. The model we propose for swiss e-ID in section 4.4 takes this aspect into account and tries to solve it by doing part of the monitoring work to the trust infrastructure.

Regarding the ”decentralized Certificate Authority” definition, one should note that in CT, domain owners make use of a CA in order to being able to affirm its own identity (the domain). In KERI we do not need certificates for that, because the identity of an entity is strongly bonded to its key and it is ”guaranteed” to some extend by the decentralized network. This means that we do not have to additionally trust a CA, and the whole chain-of-trust, as in the Web PKI. However an identity itself is not much useful. In order to make use of a KERI identity we would need verifiable credentials, such as a passport, and for this we would have issuers, which in fact are very similar to CAs.

Summarizing, we believe that KERI can be instantiated as a CT network, by loosing some of its distribution propierties. In fact, CT gives a practical example on how KERI could be instantiated. A KERI identifier does not need a CA to confirm its identity. However, certificates issued by recognized entities are needed to make use of its own identity.

4.2.2 KERI and Hyperledger Indy

Since KERI is presented as an alternative to ledger-based SSI solutions, and Hyperledger Indy is one of those solutions, we made an high level comparison of the two systems. KERI and Indy use opposite approaches. In KERI, we build a whole network in order to spread and somehow protect the public keys, or the key events, of every identifier. In Indy, this is done by keeping a distributed, but centralized ledger.

We clearly see an important trade-off. Indy, or any other SSI ledger-based approach, guarantees a consensus, meaning that it is given that each user sees the same current public key, or set of transactions, about the same identifier. This is a strong guarantee, but we pay for it with limited scalability, as PBFT consensus is expensive and does not scale well. On the other side in KERI we do not guarantee a consensus by default, meaning that it is possible that two verifiers see a different current public key, or KEL, about the same identifier. The set of watchers and witnesses help giving a *best effort* consensus, but no guarantee can be given without fixing additional constraints. Nevertheless, KERI claims to have better scalability.

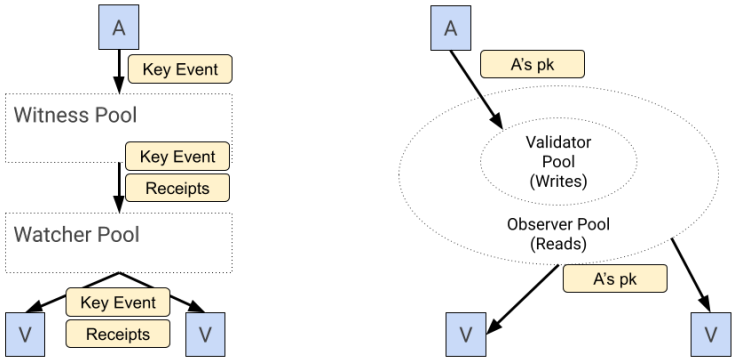

Figure 4.3: KERI (left) and CT (right). A is an identity, V a verifier. The use of the Validator and Observer pool usually require a registration (permissioned).

KERI identifiers are meant to be universal, in contrast to Indy identifiers. An Indy identity is fully determined by what it is written on the ledger. An identity in KERI is bounded to the KEL and a set of Witnesses, which can be changed. Imagine a revolution happening in the country, and people stop trusting the government. KERI gives an easy solution to that, the controller of an identifier could easily switch from an infrastructure run by the government to another one and continue using its identity. In general, KERI presents itself as an interoperable system. It could be possible to implement a similar functionality in Indy as well, but as a permissioned infrastructure by default, this could be less easy. On the other side, the permissionless approach of KERI could be harmful in some other situations.

Other differences concern the centralization and governance malleability. In KERI we have one KEL for each identifier, which can be witnessed by a set of unique witnesses. In Indy we have a single distributed ledger, distributed among a set of nodes which are usually under the same governance. On the other side KERI allows for more fine-grained modelling of governance, as described in this thesis. We are not affirming that with Indy it is not possible to model complex hierarchies, but in KERI there are different roles which can be run by different entities, whether in Indy seems like the whole infrastructure has to be under by the same governance.

A question is whether we can archive a more fine-grained governance modelling with Indy as in KERI, for example by creating a set of interconnected but under different governance Indy networks. Following the example of a federated KERI network, we could have an Indy network for a Canton, and one for the federal government. On the other side we believe that having consensus guarantees in KERI is a necessity, and this could be possible by having smaller KERI networks interconnected. In this sense, both systems could end-up in a similar configuration. It this context, KERI has the advantage of being interoperable by default.

Summarizing, KERI and Indy have opposite approaches giving different guarantees. However, we believe that both would require a similar model made of a network of networks. KERI is more interoperable whether Indy provides a consensus by default.

4.3 Weak points and limitations

In this section we would like to highlight the main weaknesses and limitations of KERI that emerged during this study. But we would also like to provide some solution or suggestion on how to deal with them.

In general we find that the concepts introduced in KERI and the roles defined in the networks are interesting and innovative. Nevertheless, using the provided building block to design an actual network is much more complicated and raises many points of discussion.

4.3.1 Duplicity and absence of consensus

Here we analyze some security aspect of KERI. In KERI there are witnesses to help protecting the controller from duplicity and watchers, in addition with judges and jurors, which should provide a duplicity protection for the verifier. We might call a watcher pool, eventually including judges, jurors and other roles, a duplicity detection network.

Let us start from a basic example. Assume there is one malicious controller C running a set of malicious witnesses, which are sufficient to witness its KEL, KEL_C. Now, assume two different duplicity detection networks exists, DD_1 and DD_2, and two verifiers V_1, V_2, which make use of one of the two duplicity detection networks. Furthermore assume the two networks have no nodes in common and no communication exists between them. However, watchers, jurors and judges in both networks are not necessarily controlled by the same entity nor governance, and could be completely independent. A trivial attack would be publishing two different versions of C's KEL, KEL_C and KEL'_C to the two different networks. This would be possible since the witnesses are as well controlled by C and could easily produce inconsistent receipts. Both duplicity detection networks would not see any issue, since the receipts would be valid and they would not see any other inconsistent version. The same holds for the verifiers relying on them. The result is in an inconsistent view for V_1 and V_2, which would accept two different versions of C's KEL. This could result for example in V_1, V_2 establishing a different current public key about C, or even accept a different set of issued credentials if we think about non establishing events in the KEL.

From our understanding, the security in KERI is based on the fact that situations like this are unlikely, because a communication between the verifiers or

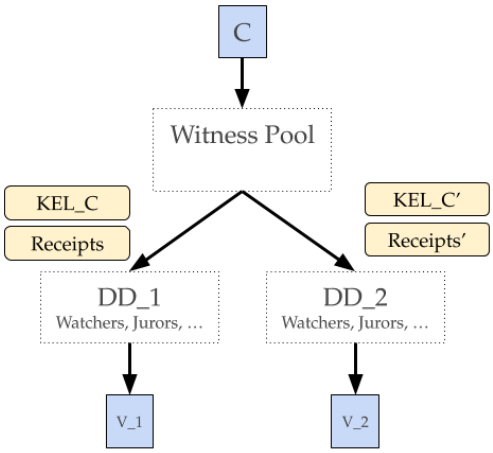

Figure 4.4: Example on how two verifiers relying on two different set of watchers (Duplicity Detection networks) could result in accepting two different versions of the same KEL. We assume no communication exists between the two set of watchers.

watchers in different sets will eventually find the inconsistency. While we think this is not necessarily true, the eventual, or best-effort consensus might not be sufficient in some situation, like for the swiss e-ID.

In the previous example we assumed that a malicious controller C was able to produce duplicity, with the help of a compromised witness. In the following example (see 4.5) we show a situation where duplicity is produced by the duplicity detection network. Assume a controller C, an independent set of witnesses, two distinct, meaning absence of communication between them, duplicity detection networks DD_1 and DD_2, where the watchers, jurors and judges in DD_2 are controlled by the same entity, we call it X. Imagine X was able to compromise the current key of C, and for this reason C decides to rotate. The witness will regularly release receipts. DD_1 will regularly propagate the update to V_1. But DD_2, under the control of X, decides not to propagate the rotation event, in order to continue impersonating C thanks to the compromised key. V_2 would not see the rotation event and eventually continues to accept the compromised key. This holds unless V_2 is able to communicate with other actors than the one in DD_2. A similar situation could happen with the same entity controlling the witnesses. Note that one might argue that a verifier can chose the watchers and the assumption there is that the verifier picks trusted watchers. However in general the verifier might not known whether the watchers are controlled by the same entity, both in the case of a compromise or for other reasons.

The assumption of no communication between the nodes, from the above examples, might see unrealistic when thinking about two sets of nodes. But imagine KERI in a large, world-wide scale. It would be impossible for all of them to exchange information about all KELs of all identifiers. This

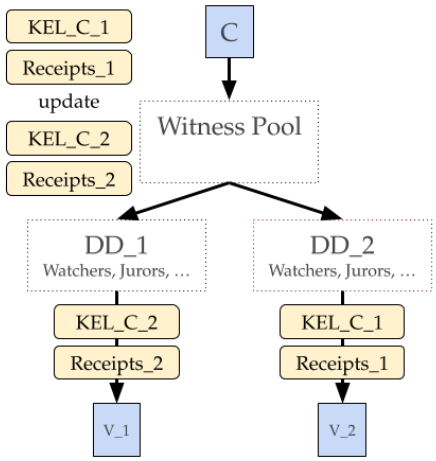

Figure 4.5: Example on how preventing an update from sharing creates duplicity. The update event (KEL_C_2) is not sent to V_2. Watchers in DD_2 are all controlled by the same entity and no communication exists between the two networs nor between the verifiers.

discussion came up during the DICE conference in Zurich and Samuel Smiths suggested[2] to introduce other roles which communicate between the different sets. One of them is called *Super Watcher*, essentially a watcher exchanging information between distinct set of watchers. However, how much is this approach scalable? Should everyone in the end trust a single entity, in the form of a *root Watcher*?

We do not believe that KERI in general is scalable and can prevent all of the presented attacks at the same time. However in this thesis we present our ideas about how KERI could achieve both properties, by adding some constraints.

4.3.2 Providing guarantees

It should be clear that in absence of consensus duplicity can always be produced. The optimistic, or best-effort, approach of KERI based on eventual exchange of information and reputation, might not be enough for many applications. Here we suggests some approach on how to provide guarantees for a KERI network.

Trusted Witnesses

A first approach would be to use a set of trusted Witnesses, and require the users to use them. In fact, this is the approach used at GLEIF, where, at the time of writing of this thesis, they do not even operate watchers. In

[2]DICE 2024, Book of proceedings https://qiqo-files.s3.eu-west-1.amazonaws.com/b4n5ryq5wnnypyz096phecyaqbuu

this context, watchers are not needed, since we can assume that witnesses never produce (intentionally) duplicity, because we trust them. In this sense KERI is still bringing some advantage. The pre-rotatated key allows post-compromise recover of control and multi-signature schemes allows to model complex hierarchies. Most importantly we can reach a byzantine consensus by running KA2CE, nothing more than byzantine fault tolerant consensus, on the trusted Witnesses. This seems sufficient to prevent duplicity produced by the controller, as it has no control over the witnesses. While preventing an event update to be shared is still possible, in this case the trust assumption might be sufficient, as the witnesses are the only infrastructure we can use, hence we have to trust it. To emphasize, the difference in the above example was that a verifier could chose another infrastructure (another duplicity detection network), and here the trusted witnesses are the core of the system.

Trusted Watchers

Following the same idea we could fix a set of trusted watchers, which we call duplicity detection network or watcher pool. The advantage here is that we offer the controller the possibility to trust any witness they prefer, which is a fundamental concept in KERI. On the other side we force a verifier to trust the provided duplicity detection network.

4.3.3 KERI as a Network of Networks

Both presented approaches introduce a trusted party, centralizing de-facto parts the full decentralized concept of KERI. However, we believe this is a necessary trade-off in order to make use of KERI.

Another issue is that we can hardly scale this approach, because this would require everyone to trust the same set of witnesses and watchers worldwide, in a same matter as in CT. It is not only a possible technical limitation, but as well a limitation on trust. Does a nation trust a third party for issuing and revoking its own national digital identities? It is probably not the case.

This is where we introduce our vision on KERI as a network of networks. A network would be a space of mutual trust and governance, made of a number of identifiers using their own witnesses and a provided, trusted duplicity detection network. Inside this network some authority could issue verifiable credentials, such an ID. When a member of a network interacts with a member of another network, a mutual trust between the two networks has to be established by using the respective duplicity detection network. This also makes sense from a legal aspect, since the network would be accountable for non detected duplicity.

As an example, imagine every nation as an independent KERI network. A citizen a from state A wants to interact with a citizen b from state B. In order

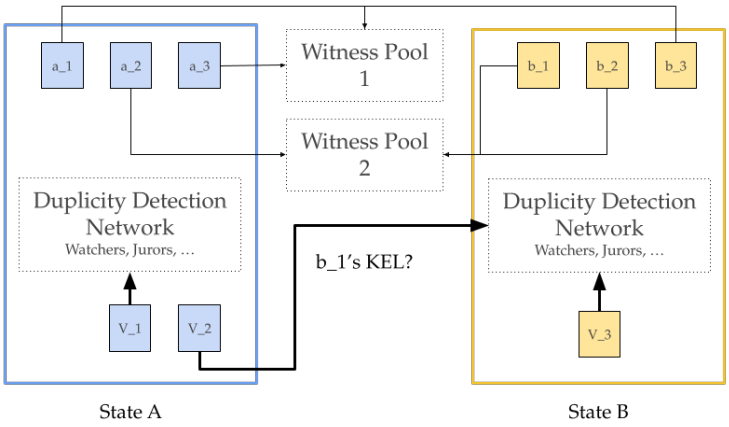

Figure 4.6: Example of two networks. Every controller can freely choose its own witnesses. V_2 relies on B's duplicity detection to fetch b_1's KEL

for the verifier a to establish the current key of b, a should trust and rely on B's duplicity detection network. This is the same trust model that we observe in reality with physical passports. If we want to extend this example, imagine country A blocking a for any reason. This is possible, since A can prevent a's event updates to propagate trough the duplicity detection network. a could ask political asylum to B just by asking other to use B's duplicity detection network. Of course, in this situation we do not care anymore about eventual credentials released by A.

This model would allow any kind of interaction, as an example the authorities of one network could release verifiable credentials to the identity of another network, as long as they trust the other network, while clearly structuring the responsibilities.

As another example, imagine a swiss e-ID network and a bank network. The banks could trust the swiss government providing a network to establish an identifier's key and an identifier's identity thanks to some issued credential (the e-ID). If for example a user was able to create duplicity and get two different e-IDs, the swiss government would be liable for this. On the other side the bank could run identifiers related to bank operations. If they use it for transactions and somehow a double-spend due to duplicity in the KERI bank network happens, the bank would be liable for it.

One could argue why we do use a decentralized system to build a network of centralized networks. In fact this is not necessarily true, because we do centralize and structure trust, but our structure do not affect the functionality of any identifier, which could still be an independent entity, free to choose its witnesses and control its own KEL. The focus here is about enforcing a structure in KERI, and making the use accountable, which we believe is the right way to make use of it.

We believe that this approach is more scalable, since it connects networks of arbitrary small size, gives guarantees inside the network or based on mutual trust and allows universal interoperability. Worth mentioning that we share the same vision on KERI as the Human Colossus Foundation team[3]. In particular, they are working on principles for a universally interoperable KERI. In this thesis we present an ad-hoc solution for the swiss e-ID, not a general KERI system. However we still want to pay attention to the interoperability of the system.

4.3.4 Privacy

The analysis of the privacy proprieties of a KERI network and ACDC credentials is beyond the scope of this thesis. However, we would like to briefly mention some aspects, as it is an actual topic.

In general KERI as it is presented has poor privacy properties. This is because the interactions in the network are all authenticated and in clear. For example, when a verifier has to query a watcher, the latter can record who is asking about who. It is claimed that since an identifier is not necessarily bounded to a physical person, profiling is not a problem. We argue that in fact an identifier without an identity, for example with a credential, is quite useless, and delegation is trackable, so not very helpful in this case.

As well ACDC credentials suffer from the same problem, they are not privacy-preserving by default. Bulk-issuance of credentials could help, but in general to archive better privacy some adaptation has to be done, in our opinion.

4.4 Proposed Model

Based on the analysis done on KERI, and the requirements given by the swiss e-ID taskforce, we propose a design model for the use in the context of the swiss e-ID. This design is the result of multiple iterations. Starting from a first proposal, we tried to improve the KERI proprieties and solve practical problems, as well as we adapted the network to fit the swiss constraints, in direct collaboration with the swiss e-ID team.

4.4.1 Design

The design takes into account the constraints for a swiss e-ID service. In particular it is required to provide a whole infrastructure, meaning a self-contained infrastructure, in order to offer a complete and functional service to the users.

Let us list all the components of the proposed infrastructure:

[3] https://dkms.colossi.network/

- A **witness pool**. This is provided optionally to any controller with free access. It is mandatory for the controllers of an identifier involved in the e-ID issuance or management, in short for all identifiers with some authority for the swiss e-ID.
- A **duplicity detection network**. This is the most important component. The network is made of a pool of watchers, which guarantee absence of duplicity. The watchers interacts with any witnesses, as defined by each KEL. A Judge takes the role of both a Judge and a Juror, as defined in KERI, and additionally takes countermeasures in the case duplicity is detected. As an example it could revoke an e-ID in the case the issuee, a citizen for which an e-ID has been issued, is acting duplicitous.
- A set of **Issuers**. Those are KERI identifiers controlled by the state. If ACDC is used, they have its own TEL. Even if other credentials formats are used, their signature is considered authoritative and they issue some sort of verifiable credential. Due to its critical function, its KEL can be additionally protected by using delegation or multi signature schemes together with other Issuers or so called Signers.
- A set of **Signers**. They can partecipate in multi signature schemes, with Issuers or any other identifier controlled by the state which might require it.
- A (general) **Controller** (A, B, C, D in figure 4.7). This is a normal user who ask for issuance of a swiss e-ID. Its KERI identifier can have any witnesses (up to a defined amount). Optionally, it can use the provided witness pool. Its tally value must accomply KA2CE.
- A **Verifier**. Any controller acting as a verifier must trust and rely on the provided duplicity detection network when establishing a KEL in the context of swiss e-ID. Any Verifier must run KA2CE to ensure a byzantine consensus and protect itself and the pool from duplicity produced by some faulty watcher.

In addition to the defined roles, we introduced some parameters:

- N_U: the number of users in the sense of identifiers
- N_I: the number of Issuers (of the e-ID)
- N_W: the number of Witnesses inside the state-provided witness pool
- S_W: the min number of Witnesses inside the state-provided witness pool that have to be trusted by a controller using this pool
- S_R: the tally value, i.e. the number of required Receipts in order to accept an event
- N_V: the number of Watchers inside the state-provided duplicity detection network

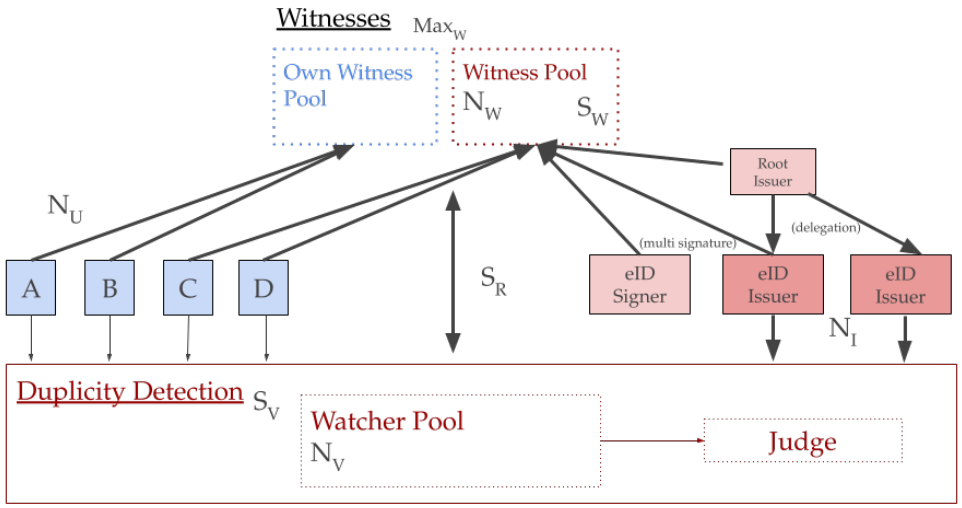

Figure 4.7: A sketch of the proposed swiss e-ID KERI network. In red is everything which will be provided and run by the state. Blue is everything the user runs.

- S_V: the tally value of Watchers receipts, i.e. the number of required Receipts in order to accept an event
- Max_w: the maximum number of witnesses specifiable by a controller. This is needed for scalability reasons (a watcher cannot query thousands of watchers, think about DoS as well)

Note that some parameter is variable depending on the controller's setting. For example, S_R depends on the number of witnesses trusted by the controller.

Excepting N_U, which should correspond to the population of Switzerland (say 9 millions, without including a growth in the foreseen future), it is hard to estimate values for the others parameters. The reason is that a KERI network like this, for the purpose of a nation-wise identity management has never been proposed before and never run in practise.

We can look at the technical requirements of vLEI at GLEIF, which is running a KERI network. In this document [8], they specify the following parameters, for both the GLEIF Root and the vLEI Issuer configurations:

- $N_W \geq 5$
- $S_W = S_R$
- S_R as per KA2CE
- $N_V \geq 3$

- S_V: 2 (when $N_V = 3$), as per KA2CE (when $N_V > 3$)

Note that at the best of our knowledge GLEIF is not (yet) running watchers. They only specify that the GLEIF Network SHOULD be protected by Watchers.

They also specify that the keys of Watchers and Witnesses MUST reside on a different device, which MAY be an HSM or a TEE. We think that this rule should also be part of the swiss e-ID Network specifications.

While those values are a good starting point, we should remember that the GLEIF network has a different use-case, as in general seems to be much less loaded as it would be the e-ID network. In order to try to give an indication of which parameters could be used in practise, we conducted a Network Evaluation experiment, presented later in this thesis.

4.4.2 Discussion

There are a few points regarding this design for which we discuss in the following, with the hope that they could help as well a KERI network design in general.

The first observation is about the method we decided to use to provide some degree of consensus. We decided to introduce a duplicity detection network, controlled and run by the state, because we believe this is the right way to deal with KERI. Also, we wanted to analyze a different scenario than the only running KERI network, the GLEIF vLEI KERI network, which uses a model of trusted witnesses. One might argue that forcing a user to trust a party, which is the state in this case, in a distributed system does not just make sense. But in fact a user will trust the state about the issued e-ID credentials, so why would it not trust the relative protection mechanism? Furthermore the user has an incentive to use the provided watchers for legal reasons: the state might be accountable for eventually undetected duplicity.

Another discussion is about who should be able to issue a certificate. We defined a specific role that is in charge of issuing e-ID certificates, the Issuer. However, any KERI controller could act as an ACDC issuer, by just keeping a TEL. We might desire to limit the issuance capabilities to the ufficial Issuers. This way, a verifier knows that every certificate obtained using the state duplicity detection network must come from the state. On the other side, this would limit the usage of this infrastructure by the industry. A good solution could be to make the issuance of certificates permissioned.

We specify a tally value for the watcher receipts (S_V), but a Verifier could in practise chose a lower one. The watchers must forward their answers to the judge, who can keep track of duplicity. If the verifier uses a proper tally value, it can detect duplicity as well, but it might just ignore it. In the case

the judge do not get enough answers about a specific query, he could make the remaining ones.

The last observation concerns the case when a controller makes use of both the state witness pool and the state watcher pool. Remember that the use of the latter is required when dealing with identities inside the e-ID network, whether the first is optional. However, most of the users are probably not aware of the underlying system and would go for the default setting, using the state provided witness pool. This is only the case if the user trusts the state, so we think it is reasonable. It is not realist to think that everyone will understand and care about the SSI principle of self-control and self-establishment. But we provide a system which can be used by those who wish it in a more independent way, by using its own witnesses. In practise we imagine that the state provides an app with predefined witnesses to trust and tally values in the case an identity is generated. An expert user could change this values in an advanced settings section, or use its own app.

4.5 Security guarantees of the proposed Model

In the last chapter we showed the exposed model, the different roles and parameters. Here we want to conduct a more rigorous analysis, which we can use to derive some security guarantees on the defined network.

4.5.1 Assumptions

4.5.2 Security Proprieties

We define the following security proprieties:

Security Property 1 *Every verifier relying on the provided e-ID Watcher Network agrees on the same KEL of any identifier*

This is probably the most important property, which ensures that two verifiers cannot accept two different public keys about the same identifier. Furthermore it ensures that every verifier agrees on the issuance or revocation state of ACDC credentials, since those are anchored in the KEL of the Issuers.

In the context of the swiss e-ID it is important to have a consistency on the status of the credentials. In particular once a credential is revoked, it must no longer be accepted by any verifier. The property ensures a consistent view. It is also important that a user can only use a single key at time. This is to prevent attacks and encourage non-repudiation.

Security Property 2 *Only the controller of an identifier can add a key event to its KEL*

This is relevant for the e-ID as it represents to the capability of a user, or a citizen in this specific case, to control its own digital identity. Even if the

state is running the infrastructure, only the user is able to change its current authoritative key, or to change its trust on the witnesses. Note that interaction events are not covered by this statement.

Security Property 3 *Sufficient witnesses inside the provided witness pool are always available to validate a KEL event. Once the controller collected all the necessary receipts from the witnesses, all the verifiers relying on the e-ID watcher pool agree on the same updated KEL*

This property completes SP2, since it states that it is always possible to recover from a compromise. This is because a rotation, which is a KEL event, used for changing its possibly compromised current key, has to be accepted. Here we state that it is always possible, when using the provided infrastructure. Furthermore, since Issuers use the state provided witnesses, the rule ensures as well that we can issue or revoke ACDC credentials at any time, since they are finally anchored in the KEL.

4.5.3 Assumptions

The following assumptions are needed to prove the security properties. They are part of the specifications of the swiss e-ID network.

Assumption 1 *At most F_W witnessess out of N trusted by a controller are faulty*

Assumption 2 *At most F_V watchers out of N_V inside the state provided pool are faulty*

Assumption 3 *A controller must set the tally value for witness receipts S_R as per KA2CE and F_W*

Assumption 4 *A verifier always rely on the duplicity detection network and accept events only with sufficient watchers receipts according to the tally value S_V as in KA2CE and F_V*

Assumption 5 *An attacker cannot compromise the pre-rotated key of any controller*

In assumptions 1 and 2 we use the term faulty. With this term we mean an entity which in general does not behave as it is supposed to do on the protocol. It could be because it is compromised, malicious or even unresponsive or unreachable.

Assumptions 3 and 4 are related to byzantine guarantees. In fact we assume some maximum number of faulty entities, and based on the byzantine fault tolerance algorithm KA2CE we can then confirm the security requirements.

Assumption 5 is fundamental in KERI. It is still reasonable because we can keep the pre-rotated key in a secure or trusted hardware, without exposing it before needed for rotation.

4.5.4 Proofs

In the following we prove the soundness of the presented security properties, given the assumptions. While all the proofs are quite trivial, making them explicit also allow us to add some additional remark.

SP1

Assume there are two verifiers relying on the e-ID Watcher Network who see two different versions (two copies of the same length, but with different entries) of the same KEL. It could be because they queried different subsets of watchers. This is a contradiction, since assumption 4 guarantees a fault tolerant byzantine consensus. It could be because more watchers are faulty. This is a contradiction, since assumption 2 guarantees an upper bound on the faulty watchers. Since the two verifiers relies on the same set of watchers, there are no other possibilities.

SP2

This follows straightforward from assumption 5, since only the owner of the pre-rotated key is able to issue a rotation event.

Please note that this is not true when talking about interaction events, which are suggested to be used as events to anchor the TEL entries to the KEL. So this could be a security risk for the Issuers. However, we could make use of multi signature schemes to enhance security in these situations. In alternative, it is always possible to anchor TEL entries by using only rotation events. It is not clear whether it is more inefficient or less secure, due to the fact the Issuer should frequently make use of the pre-rotated keys.

SP3

Assumption 1 and 3 guarantees that there is always a sufficient amount of non-faulty witnesses in order to collect a sufficient number of receipts to validate its own KEL events. Once the controllers has collected the receipts, they must be already available for any watcher. The same amount of non-faulty witnesses can provide sufficient receipts to the watchers. Because of SR1 we guarantee that the update is transactional and that every verifier agrees on that.

Note how assumption 1 and 3 are not specific to the e-ID witnesses pool. In general, if the controller set proper tally values, we can archive byzantine fault tolerance.

4.6 Meeting the e-ID requirements

After presenting the proposed KERI network we want to analyze how this solution does fit the swiss e-ID requirements.

4.6.1 Design principles

Let us address each point specified in the discussion paper [10]:

Privacy by design

- *The e-ID Act follows the SSI principles with three different roles: issuers, holders, and verifiers. These roles communicate directly with each other.*

  This is met by design by KERI.

- *The issuer has no knowledge if or how the verifiable credentials that it issues are used. The e-ID Act defines this privacy-preserving principle.*

  If we use ACDC: it depends on the TEL implementation. The TEL does not have to be provided by the issuer, and we might assume a privacy preserving inclusion queries on the TEL. The issuer should only anchor updates to its KEL in order to sign the TEL. However, only checking the last KEL anchor is sufficient to authenticate the whole TEL, which is cryptographically chained. This operation is independent from the credentials.

  If we use other VC formats and systems: it depends on the used VC.

- *The system should protect privacy by default. (...)*

  Even if privacy was not the main focus of this thesis, we can affirm that KERI does not address privacy issues. KERI provides a base for key establishment. Clearly, each role could record and correlate KERI queries. This do not fulfill unlinkability, which is a requirement of scenario B as explained in the discussion paper[4]. Furthermore default there is no encryption mechanism, but queries, KEL and TEL do not contain any personal information. A KERI identifier is meaningless without an authentic issued credential.

Data minimisation

- *A verifiable credential can be presented in its entirety or only partially (selective disclosure);*

[4] https://github.com/e-id-admin/open-source-community/blob/main/discussion-paper-tech-proposal/discussion-paper-tech-proposal.md#a3unlinkability

ACDC supports selective disclosure. But we can use any other VC format supporting selective disclosure on the top of KERI.

- *If feasible, information can be presented as a derivation (predicate proof; e.g., "is older than 18") or as a "zero knowledge proof".*

  This depends on how credentials are made and on its content.

- *The necessary central base registry only contains the minimum of data concerning registered issuers and verifiers or related to revoked credentials. There are no traces of data related to individually issued credentials stored centrally. The e-ID Act defines what information is contained in the base registry.*

  KERI do not expose any personal information, nor any credential information when using ACDC and TEL. Only an hash of the credentials is exposed in the TEL, in order to check its status (issued or revoked).

- *Unnecessary data flows are avoided through the SSI paradigm (e.g. by using direct communication between the concerned actors).*

  This is more a concern of systems in the top of KERI. KERI requires necessary communication in order to establish the current authoritative key of an identifier. CESR, the binary encoding used in KERI is designed with efficiency and data transmission minimisation in mind.

Decentralised data storage

It is achieved by storing the e-ID and other verifiable credentials solely in the user's wallet. From there, it is presented to a verifier. As a consequence, every transmission of data is under the control of the user, and the presentation of data to a verifier always requires the active consent of the user.

No credential data is stored in the KERI network or by KERI actors.

4.6.2 Legal requirements for the base registry

This is strictly related to the KERI system. Let us enumerate the various points written in the law [3], Art. 2 (translated):

1. *The Federal Office of Information Technology and Telecommunications (FOITT) shall make available a publicly accessible basic register; this shall contain data that is required:*

   a *to verify whether electronic evidence such as cryptographic keys and identifiers have been subsequently modified;*

   This is exactly the scope of KERI. We believe that together with the proposed network, modifications are guaranteed to be shared and verifiable

b *to check whether the electronic evidence originates from the issuers entered in the basic register and the corresponding identifiers;*

This is done with signature verification based on the key established with the KERI network.

c *to enter persons in the trust register who issue (issuers) or verify (verifiers) electronic evidence;*

KERI provides an identifier, which can be used by the trust register.

d *to check whether an electronic certificate has been revoked.*

This is possible with KERI ACDC credentials. Also any other VC credential with a revocation check method could be used.

2. *Issuers and verifiers may enter their data in the basic register.*

Each identifier in KERI, including issuers and verifiers, can change and control its KEL. In the proposed model they have complete freedom on their witnesses.

3. *The basic register does not contain any data on the individual electronic certificates, with the exception of data on their revocation.*

This is met in KERI and with ACDC as well.

4. *The data on the revocation of electronic certificates must not allow any conclusions to be drawn about the identity of the holder or the content of the certificate*

Revocations are contained in the TEL. Only hashes of the credentials are exposed, which cannot reveal any information about the holder or the content of the credential.

Paragraph 5 regulates how data generated during queries of the base register may be legally used.

4.6.3 Recommendation

The presented KERI network only partially satisfies both the legal and technical requirements for the base registry of swiss e-ID. In particular, we distinguish between the two proposed scenarios. While KERI satisfies the requirements for scenario A, it does not meet the more privacy preserving demands of scenario B. The main problem here is the linkability of KERI identifiers.

In order to meet the described security proprieties we had to introduce more complexity in an already complex system. We think that the complexity of KERI is not justified by the mean, in the use-case of the swiss e-ID. In particular with KERI we would design a fully distributed system for a fully

centralized governance such as the issuance of e-ID credentials. While many concepts are innovative, such as the pre rotation feature, our opinion is that it is not suitable for the swiss e-ID.

This is our recommendation on a theoretical side. More insights about practical feasibility are give in the section 5.2.4.

Chapter 5

Network Evaluation

In this chapter of the thesis we present the setup and the results of the KERI network evaluation experiment that we conducted. The aim of this evaluation is primarily to assess the feasibility of running a large KERI network in the wild based on the existing libraries. Secondarily we would like to evaluate scalability parameters to support a better estimate of the values which could be used in a swiss e-ID network as suggested and defined in this thesis. In addition to that we contribute to the development of the keriox library.

Excepting the KERI deployment at GLEIF, we have no knowledge of any other effort made to run a large KERI network or any KERI network with watchers implemented. This experiment is the first large scale KERI network experiment with watchers, to the best of our knowledge.

The whole code and the results are available on this github repository: https://github.com/luffa99/KERI-Under-Scrutiny-Master-Thesis

5.1 Implementation and Setup

Remember that our network evaluation is based on the swiss e-ID KERI Network model presented in this thesis. Furthermore in our experiments we require users to trust a the provided witnesses and all of the watchers.

5.1.1 Choice of library

For this evaluation we decided to use the keriox library, which implements the KERI core functionalities in rust. The decision was driven by two aspects. The first is that KERIpy has a team which is currently working on it and this library is used for the deployment at GLEIF. We wanted to give a chance and our contribution to the alternative implementation in rust, which has a smaller team of developers. Furthermore KERIpy is oriented for the use-case at GLEIF, while keriox is based on the principle of universal usage. The

second aspect concers the implementation of watchers, which is still missing in KERIpy. According to our considerations exposed in this thesis, we want to test how watchers behave in a KERI, so keriox was the only possible choice in this sense.

Keriox

KERIox is a library developed at the Human Colossus Foundation and released under the EUPL1.2 license[1]. The non-profit foundation based in Geneva operates in the field of governance and digital data economy. In this context the foundation not only researches solutions such as open standards, but works on them as well. This is the case for the keriox library.

We are not aware of previous use of this library to run a KERI network with a large amount of users. In fact the library is still work in progress and we can be defined as the first real users of it. For this reason this thesis contributed as well to the development of keriox. In particular during the implementation of our test infrastructure, we identified many and different types of issues and bugs. A close collaboration with the keriox mainteiners and in particular with Edyta Pawlak and Michał Pietrus, allowed for a quick fix of the problems and an improvement of the library.

To give an idea, the version of the library at the begin of this thesis while correctly implementing KERI functionalities, was not working with more than 3 users. Thank to our contribution keriox can now be used to run a distributed network with thousands of users. It is important to specify that the library do not has yet a convenient API for operations with it. For this reason we had to write some wrapper code in order to use it more easily. Since it is still a work in progress, we also had to adapt multiple times our code to the fixes that have been done in the meanwhile.

5.1.2 Concepts in practice

After understanding how KERI works on paper, it is time to translate the KERI concepts in code. Independently from any library, we realized that not all the concepts proposed about KERI are easy to implement. Think about the role of watchers, which are defined as entities watching over identifiers. How is this done in practice? Should a watcher query all the witnesses every second? What if the witnesses to query are millions? What if an attack happens within this second? We obviously need some compromise here.

[1] https://eupl.eu/1.2/en/

Verification in practice

We decided that in our implementation each verification has to follow particular steps. These steps are mainly determined by the keriox implementation. In particular, a verifier should:

1. Have an event seal to verify. An event seal is a reference to a position in the KEL of a verifier, and is composed by an identifier, a sequence number and a digest of the event.
2. Query the (state provided) watchers.
3. The watchers answer the query if they have a cached version of the KEL corresponding to the provided event seal provided by the verifier, otherwise they forward the query to witnesses

An alternative to using the event seal would be to always query for the last event in the KEL. Since this possibility was not yet implemented in keriox, we decided to follow the approach explained here.

Note that this is a large difference compared to the model in theory. The model requires that each verification query goes up to the witnesses and fetches the last event in the KEL. In keriox, the event seal has to be passed to the watchers, which compare it against a local cached KEL. If the event is present, they will just answer with a local copy and would not detect an eventual update. This is a difference in both security and scalability. Security, because an attacker could continue to use the event seal of the key it has compromised, and as long one of the watchers do not receive from anyone else a more updated event seal, it will continue to verify the event. Of course looking up in the cache instead of querying multiple witnessess is a difference in terms of resources and hence scalability.

Keeping the KEL and verification with old key

With an eye on scalability we would like to discuss how a KEL should be kept in practice by a watcher and or a witness. Assuming a huge numbers of identifiers, and KELs with tausends of entries, keeping the whole KEL for each identifier does not seem like a scalable and storage efficient solution. KERI claims that it is possible to just keep the last entry of the KEL in order to verify the last update. This is not yet implemented in keriox, so is not present in our implementation as well. However we see a problem in this approach. In particular, this would not allow to verify signatures using an old key. While this might make less sense, imagine an issuer changing its key. We should not consider all previously issued credentials as not valid anymore because of the change of signature. In KERI ACDC this problem is solved with using a TEL. But in general it would be hard to verify older message, if it is necessary. In any case, the controller could provide the entire

KEL in the case where this would be necessary, assuming those cases are quite limited.

5.1.3 Implementation Framework

As mentioned above, keriox does not provide a product ready to be used. For this reason in the context of this thesis we wrote a framework to run KERI. The framework was necessary to run the roles of controller, witness and watcher in practise on an infrastructure and to collect the data used for the analysis. The code was written in both python and rust, and takes care of the entire pipeline, from compilation to execution on the infrastructure.

In particular an entire wrapping code was produced to have a convenient API for the KERI basic operations, like initiating an identifier, verification of a signature, verification of an ACDC credential, rotation, etc. Keriox only provides some of these operations as API. Furthermore we implemented features such as exponential back-off re-transmission for all the KERI communications, which we consider fundamental for the use in practice. Watchers may need some time to fetch the necessary receipts, or they might be overloaded. For this reason not all queries are always successful and this is why re-transmission at application level is needed.

Furthermore we wrote specific code to run tests. A python script compiles the rust code and collects and sets up the settings files. Then, it spawns virtual machines within the amazon AWS network and manages the entire communication to transfer the data, start the tests and collect the results. We wrote two rust clients. The first runs on the local machine and coordinate the tests. In particular it synchronizes the start and the end of the tests with all the test clients. This synchronization allows us to create a decent amount of load on the KERI network at the same time. It takes as well the role of the issuer. The second client runs on amazon AWS and has the role of controller of an identifier and verifier of the issuer.

5.1.4 The infrastructure

We relied on Amazon Web Services (AWS) to run the infrastructure on many different EC2 containers. The latter, also known as Amazon Elastic Compute Cloud, offers flexible on-demand computation power over virtual machines. We decided to use the following instance types:

- t3.micro with 2 vCPU and 1 GiB of RAM, for running users (controllers of a KERI identity, excluding witnesses and watchers)
- c5a.2xlarge with 8 vCPU and 16 GiB of RAM, for running watchers and witnesses

The virtual machines run over different physical machines in different locations. We decided to use the following locations in Europe: Frankfurt (eu-central-1), Ireland (eu-west-1), London (eu-west-2), Paris (eu-west-3), Stockholm (eu-north-1), Zurich (eu-central-2), Spain (eu-south-2) and Milan (eu-south-1). The use of different locations ensures that the virtual machines are actually running over different physical machines and introduces realistic communication delays for a distributed network. Since an upper limit on the virtual machines is set by AWS, we had to request a quota increase to be able to run more users.

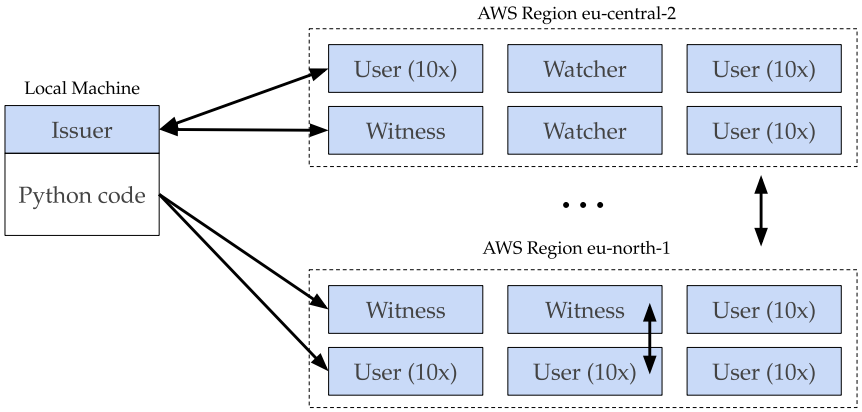

Figure 5.1: Overview of the setup. On the left we have the local machine, including the python code to spawn, manage and syncronize AWS VMs and the issuer, which communicates with witnessess and users (for example, to send a signed message). On the right, we have many VMs in different physical locations (AWS Regions), with different roles. Communication occurs both within and between regions, as witnesses and watchers are spread over different regions.

Our client code runs up to 10 test clients (KERI users) on each virtual machine in parallel.

The total costs for running the tests, including the development and initial parameter tuning was of about 170 USD. Note that as new users we also used some free quota from the free tier.

5.2 Experiments

The client running on the local machine acts as issuer, while a variable number of clients run on amazon AWS and simulate up to 10 users each. Here is a list of the conducted experiments:

1. Inception. The issuer and all the users make inception (setup of KEL, witnesses and watchers).

2. First verification. The issuer signs a message, send it to all the users

and the users have to verifiy the signature.

3. Second verification. The issuer rotates its key, sends another message. The users have to verify the signature with the rotated key.
4. Rotation. Every user rotates its own key.

We consider these tests as significant for the following reasons. With test 1 we simulate the introduction in the network of many users at the same time, as it could happen when e-ID is launched. With test 2 and 3 we execute the main task of KERI: key establishment. Test 4 makes use of witnessess and watchers for updating the KEL of many users at the same time, without the setup phase required in test 1. Note that test 2 is straightforward given test 1, because the user collects the verification data during inception, and test 2 its only a lookup in its local database, confirming the correctness of the collected key. For this reason we do not measure the execution time of test 2, but we count it in the case it fails.

We were interested as well in testing the ACDC credential issuance and revocation. Unfortunately, the maturity of the keriox library at the moment of the execution of the experiments was not sufficient to run those tests smoothly. However, note that in general the TEL could be provided by a service external to the KERI network, and its anchor in the KEL is verifiable in the exact same way we fetch the rotated key in test 3. So we expect test 3 to cover ACDC/TEL capabilities as well.

5.2.1 Parameters and measurements

Following the definition of parameters presented along with the model in section 4.4, we require all the users to use the same set of witnesses, provided by the state, with cardinality N_W. Furthermore we set $S_W = N_W$, meaning that all the users trusts all the witnesses in the set. The latter makes sense, since the witnesses are controlled by the same entity in this case.

In the tests we variate the number of users N_U, the thresholds and total number of witnesses S_R, N_W and watchers S_V, N_V, while we only have 1 issuer N_I.

Based on a first investigation and the parameters in use at GLEIF we define a baseline for the security parameters as follows:

- 4 witnesses, at lest 3 receipts ($N_W = 4$, $S_R = 3$)
- 4 watchers, at least 3 receipts ($N_V = 4$, $S_V = 3$)

This baseline allows 1 faulty witness and 1 faulty watcher. An additional parameters is the number of users running on each virtual machine. We decided to use 10 as value. The latter was empirically determined as a sufficiently high number without affecting to much the machine's performance.

In the beginning, a number of 50 users per virtual machine was set, but we realized from the collected data that there was a bottleneck on the users machines, causing longer execution times.

First we evaluate the baseline on different number of users, then we vary parameters to see how witnesses and watchers respond.

In every test we did collect a lot of data. This data includes:

- CPU and memory load (in percentage) of each virtual machine. This helps us to understand how each roles is loaded, and where the bottleneck is. It also give and indication of the hardware limitations and or scalability requirements.
- Total amount of data received and sent on the network. This is in particular useful for witnesses and watchers to observe eventual imbalances of traffic between them.
- Timing required to execute each test. This data is collected individually for each user in every virtual machine. It is the main indicator for scalability issues. This is because it is proportional to the overload of witnesses, watchers and the protocol.
- Success of the test for each user. A user's run is considered successful if in test 1 it is able to finish the inception including the collection of witnesses receipts, in test 2 is able to verify the signed message, in test 3 is able to verify the signed message after the key rotation and in test 4 is able to complete a key rotation including the collection of witnesses receipts.

The load and traffic data is sampled every 5 seconds with a bash script. The operation timings is measured from each rust client and excluding the synchronization process before and after each test.

It is also important to mention that not all the users always succeed every test. Whenever it is the case, the timing data related to the user is excluded from the statistics.

Each experiment is executed two times and we consider average values from the two. In the case where one of the experiments was clearly an outlier, we repeated the experiment one additional time and excluded the worst one.

5.2.2 Results and discussion

Here we present the most important results and discuss them. Excluding the parameters tuning and exploration, in total we executed 15 tests, each of them twice.

Baseline

We start by executing the baseline parameters $N_W = 4$, $S_R = 3$, $N_V = 4$, $S_V = 3$ (In brief: 3/4 Wit, 3/4 Wat) with different number of users N_U. We sampled the execution times of tests 1,3,4 and the overall success rate with 500, 1000, 2500, 400, 5000, 7500 and 10'000. The results are summarized on figure 5.2. Each execution had a success rate greater than 99%, with the exception of 10'000 which had a success rate greater than 96%.

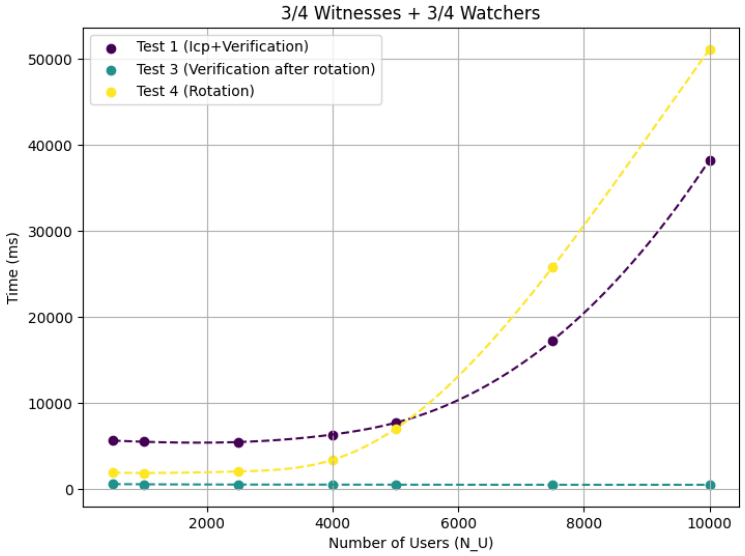

Figure 5.2: A plot of the execution times of each test with different number of users. A spiline shows the trend.

The first observation is about the timing of the operations for $N_U \leq 5000$. In this case both the setup phase (test 1) and the rotation (test 4) required less than 10 seconds. This seems a reasonable time for the use in production. We also observe that the verification (test 3), which includes collecting the KEL of the issuer, has constant time. This was expected, since the verification in keriox is not always complete, in the sense that the watcher do not always query the witnesses for updates, but can rely on its local cache. This is different from the proposed model, where for security reasons we require the watchers to always check for the last version of the KEL. However the capabilities of the keriox library did not allow us to test this behavior, which would have been very interesting to observe. We can only say that the watchers are capable to answer with cached KEL information quite fast.

For $N_U > 5000$ we observe an exponential trend, which suggests a bottleneck somewhere in the network. To further investigate this, we can have a look at figures 5.3, 5.4 and 5.5. They show the CPU and memory load of the various

actors in the network. The CPU load value can give us an idea of overloading of the machine. They collect the nodes data from the startup to the shutdown, this is why the graphs are not horizontally aligned. However, vertically grey lines are provided as reference in time: they represent in order the start of users inception, the issuer's rotation and start of the verification (note that these two can overlap) and the start of the rotation of the users. We can now explain the peaks. During inception both witnesses and watchers are involved: witnesses because of the receipts, watchers because they have to be setup by every user. During verification after the issuer's rotation we see that watchers are working, since they have to provide the required, updated KEL. However witnesses are not, because as already mentioned, in this case, watchers will provide a cached copy, after one query to the witnesses. Finally, during the overall rotation, witnesses are again under pressure, because they are required to produce receipts.

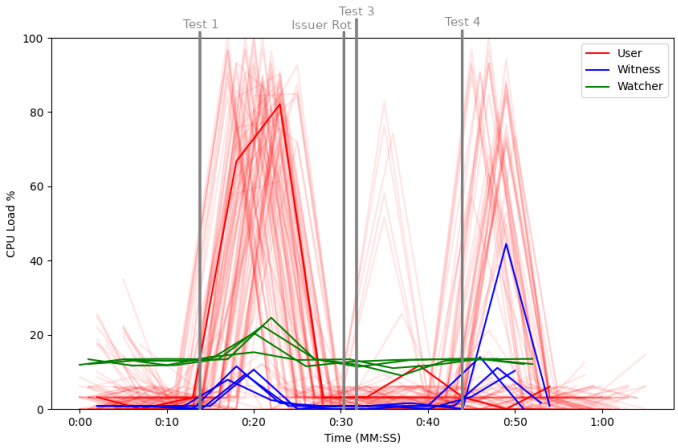

Figure 5.3: CPU load of the baseline with 1000 users

Coming back on the exponential trend, the CPU load with 5000 and 10'000 users suggests that witnesses (and watchers) are overloaded during inception and rotation, which may be the reason of such time explosion. In the swiss e-ID an initial setup time of less than 60s, as we experience here with 10'000 parallel users, might still be acceptable. The same might hold for rotation, since it is never a critical operation, even after a possible compromise. For sure, it is better to rotate as fast as possible in the case of a compromise, but remember that if you rotate for this reason, your key is already compromised and attacks are possible in the meanwhile.

We can try to extend the witnesses overload observed here to the scenario in where watchers always do a complete query, which was not possible to test. The number of requests would actually be very similar. Here in test 4, each user asks for the necessary threshold of witnesses receipts to confirm

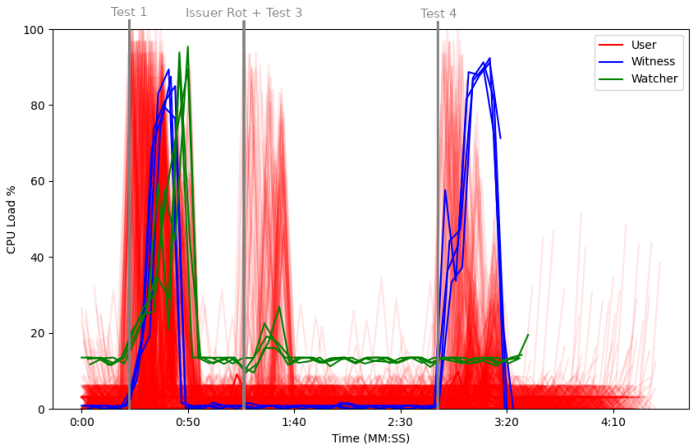

Figure 5.4: CPU load of the baseline with 5000 users

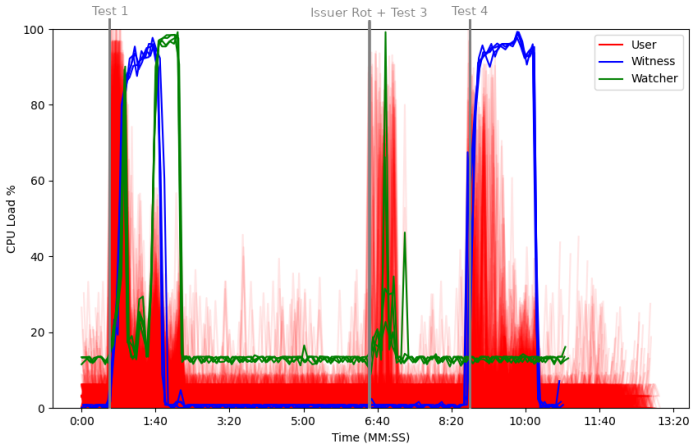

Figure 5.5: CPU load of the baseline with 10'000 users

a rotation, while in the other case watchers would asks in behalf of every user for the necessary threshold of witnesses receipts for verification. In both cases the witness should check the KEL and provide a receipt, but without having to sign it again when it is used for verification. So we would expect a similar behaviour for an entire verification as here, a bottleneck after some number of users. This is what concerns us the most, because in a swiss e-ID production network thousands of verifications would happen at the same time, and need to be fast. Even 10s would be quite slow for this application. We believe that a better hardware for witnesses would increase the number of users causing the bottleneck. However, we are unable to give an estimate of such value. Another approach would be to provide more witnesses, or more witnesses pool and randomly assign those to verifiers.

A last observation is about the rotation (test 4) taking more time than in-

ception. We expected the inception to take more time, because it also has a setup phase involving watchers. One plausible explanation is that because of this additional setup, requests to the witnesses are better spread in time during inception. On the other side, the receipts requests during rotation could be more concentrated in time, causing an overload and triggering the exponential backoff timer, resulting in longer execution times. To support this scenario, we observe figure 5.6 which plots the frequency of execution times of test 4 in one experiment with 10'000 users. Here we observe a first peak, followed by a lower wave later in time, which is compatible with our explanation.

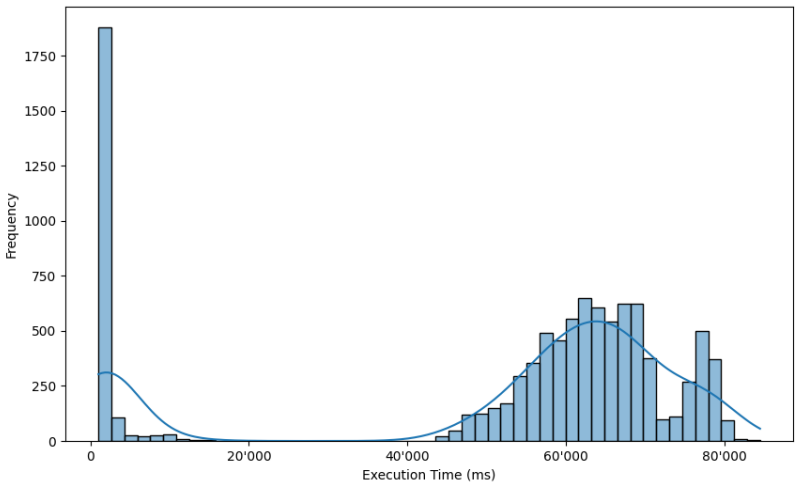

Figure 5.6: Distribution of execution times of test 4 in the baseline with 10'000 users

More witnesses

Given this baseline which works with a load of 10'000 simultaneous users, let us experiment different values for the witnesses. The indication of overload from the first experiments may suggest to increase the number of witnesses, to better spread the load. However we require a byzantine fault-tolerant agreement (KA2CE) and this is intuitively not scalable. We test as well settings with an higher value of fault-tolerance than the baseline (3/4 witnesses) tolerating only 1 faulty witness. We fix the number of users to 1000 ($N_U = 1000$) and the watchers to 3/4. Then, we test the following settings:

- 5/7 Witnesses, with fault-tolerance of 2
- 6/8 Witnesses, with fault-tolerance of 2
- 7/10 Witnesses, with fault-tolerance of 3
- 8/12 Witnesses, with fault-tolerance of 3

Here as well we had a success rate always greater than 96%.

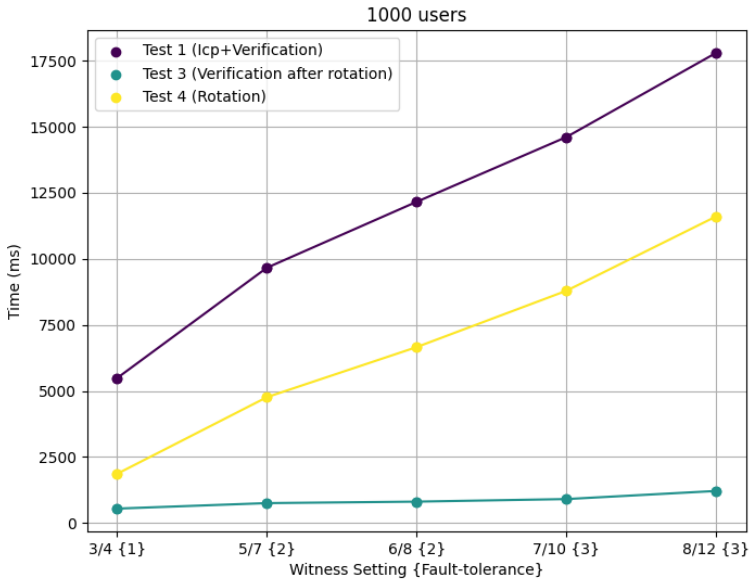

Figure 5.7: Here we have the execution time of the same tests over different witness configurations. The number of users is 1000.

As we can see on figure 5.7 it seems like, in general, more witnesses have to be queried, more the overall execution time increases. The trend is linear. The only exception here is again test 3, but remember that in this implementation the watchers will only query the witnesses once. The good news is that the trend looks linear, as expected. In our opinion, given that a swiss e-ID system should be as fast as possible, the fastest choice supporting only 1 faulty witness can be optimal. We are again worried about the timing of the verification, which we could not test.

More watchers

After studying the different latencies with more witnesses, we decided to try as well to introduce more watchers. The setting is the same, 1000 users ($N_U = 1000$) and the watchers to 3/4. We also tested the same ratios:

- 5/7 Watchers, with fault-tolerance of 2
- 6/8 Watchers, with fault-tolerance of 2
- 7/10 Watchers, with fault-tolerance of 3
- 8/12 Watchers, with fault-tolerance of 3

Here we had a success rate of 100%.

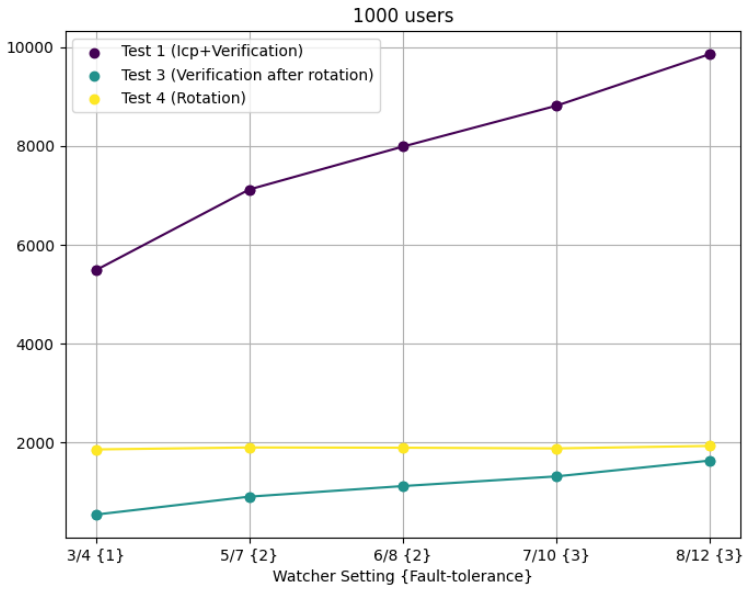

Figure 5.8: Here we have the execution time of the same tests over different witness configurations. The number of users is 1000.

The results, showed on figure 5.8, are as expected. The inception takes a bit longer as the users have more watchers to setup. The rotation is not affected, as the time here is determined by the number of witnesses. Of course, the number of watchers would affect a complete verification. Here we observe a very small increase in the time of verification (test 3), because watchers are fast answering with the cached KEL, but still we observe an increase because users have to query more watchers. If we imagine more watchers each of them making queries to the witnesses, we believe a congestion in the network is easy to produce.

While this test do not give us many insights about an optimal number of watchers, we confirmed that no issues emerged with more watchers when using keriox.

5.2.3 Maturity of library and contribution

We can affirm that we were the first users of the keriox library and we contributed to its development. While most of the KERI basic functionalities are provided, at the beginning of this thesis many of them were not even working with more than two users at the same time. The effort spent in writing a test network for this thesis included a good portion of time of discussions with the keriox developers, helping them to identify and fix the problems. Our contribution led to more than 30 fixing commits in the keriox library. Most of them were fixes of a bug or an issue we identified during the

run or setup of an experiment.

Keriox is a library still in developement and it is not ready for the use in production now. Most functionalities are there, but are not convenient to use, because a stable API is still missing. However, keriox is the only library providing an implementation for watchers, a fundamental role for protecting the verifier from duplicity. We did not evaluate the maturity of keripy, but we know that the KERI Network at GLEIF is based on it.

The keriox library is work-in-progress, and a team leaded by the HCF foundation is improving it, also thanks to our contribution.

5.2.4 Recommendation

We observed in both cases that augmenting the number of witness and the number of watchers led to a linear increase of the execution time of different tests. For this reason we recommend to keep both numbers as lower as possible, considering the number of faulty entities that can be tolerated. We cannot say how many fault tolerant entities we should consider for the swiss e-ID network, as this would require a more broad security assessment, including the infrastructure.

However we can make a recommendation on the witness pool. In particular, since the witnesses are a bottleneck, multiple witnesses pool should be provided by the swiss e-ID network. This would not affect security, which is guaranteed by the watchers.

It is true that we are testing inception (test 1) and rotation of every user (test 4) with 1000 concurrent users, while in reality people participating in the swiss e-ID would probably not generate this load on the witnesses as they would not register nor rotate all at the same time. However, since as exaplined above we can extend test 4 to the case when watchers do a complete query for verification, this load on the witnesses may be realistic (1000 concurrent verifications of a swiss e-ID).

We should reason as well on the watchers policy, and in particular we should think about relaxing the requirements. As mentioned above, it could be acceptable for a rotation to take some time before being accepted. This would mean that for some period of time (which should be defined) inconsistency can exists inside the network. But on the other side we would allow watchers to answer with their local cache, as it was successfully done in this experiment. We believe this is sufficient as long as we will use KERI for key establishment or for non time critical credentials, which could be the case for e-IDs.

Chapter 6

Conclusion

In this thesis we reported KERI's core concepts and the state-of-the-art of its extensions and libraries. We understand that KERI is mainly a DKMS, which offers a potential for an identity system with ACDC credentials. We hope that our explanation can help in understanding KERI, also because few resources and no others studies are publicly available about it.

In this paper we present the weaknesses we were able to identify in KERI, and in particular about duplicity and scalability. But we also shared how we think these can be solved by proposing a vision on KERI as a network of networks. Due to the nature of the system, we understood that many interpretations and applications are possible. In addition, no formal security analysis has ever been presented, making a security assessment quite difficult.

We deduced that KERI is in general not suitable for short-lived transactions with strong real-time requirements, as for example would be a bank transaction. The reason is that KEL updates may require time to propagate, or may not propagate at all in some attack scenarios. This may not be desirable even for non critical identities. On the other side we consider the idea of pre-rotation interesting and we think that should be implemented in other systems as well.

We think that KERI may be good as a PKI which includes attack detection, as in CT. In general KERI seems more suitable for managing a limited set of identifiers which are more important and do not change frequently. This justifies the effort for providing recovery from compromise and makes the proprieties of the network sufficient. We would not need real-time verification, and the longer propagation time would be fine for this use-case. The use at GLEIF is such an example.

The insights gained during the analysis led us to present a model of a KERI Network for the swiss e-ID, a first concrete proposal on how to use KERI for a national-scale identity system. We found that this model only partially

fulfills the legal and the technical requirements of the swiss e-ID. In particular there are privacy issues concerning linkability of KERI identifiers. We fixed policies and parameters in order to give security guarantees to the network. We presented for the first time a limited security analysis on a KERI network, including well-defined requirements and proofs.

Finally we decided to run KERI in practice on a real network with up to 10'000 users on more than 1000 virtual machines in different physical locations around Europe. No one else ever tried to run KERI, with both witnessess and watchers, on this scale before. We showed that it is possible to run such a network with the keriox library, but we identified some bottlenecks and limitations. However it is hard to predict the behaviour in practice, which could be better evaluated by running a production sandbox. We assessed the maturity of keriox, which we do not consider ready for production. Furthermore, we contributed to the development of this library by helping finding bugs and issues. Our help led to more than 30 commits. The results of this evaluation led to recommendations for the swiss e-ID network.

We are aware of the limitations of this study, especially because of the complexity of KERI and the many possible uses. However we believe that this thesis can help people to better understand what KERI is and what it is not, and whether it can be used for the swiss e-ID or for a national identity system in general. We also hope that this first analysis starts a discussion on KERI, so that we can determine if this is the system we need as the future trust system of the internet.

6.1 Limitations and Further Work

In this study we have analyzed both KERI in general and KERI as a network for the swiss e-ID. We focused the analysis of security aspects for our use-case, also because it is hard to reason about KERI without instantiating a network. It could be interesting to analyze the security of the use-case of KERI as a new web PKI.

A probabilistic security analysis could as well be interesting. For example, if we think about the proposed model, issuers may be more protected, since its KEL is fetched more frequently. On the other side, holders are less protected as their KEL is fetched more rarely. This may be a new way to see security, but further study on this is needed.

The privacy aspect could be further investigated. We only affirmed that KERI identifiers are linkable. We did not reason about which information a witness or watcher is able to obtain about an identifier, by correlating queries. This is still an open question.

Concerning the network evaluation phase, our main limitation was related to the watchers behaviour, which were not making complete queries on each verification request. This limitation is inherited by using the keriox library. To make a complete assessment on our model, a running test about this would be required. Of course, many other tests could be run and with different computation powers.

6.2 KERI for swiss e-ID

6.2.1 Base Registry

In conclusion we think that KERI does not completely satisfies the legal and technical requirements, as presented in section 4.6. The main issue has to do with linkability of KERI credentials, which does not meet the higher privacy protection demands of one of the implementation scenarios. We presented a model in section 4.4. We consider this model rather complex and we had to introduce a special policy in order to guarantee a set of security properties. Thanks to the experimental network evaluation we give some recommendation on the parameters to use in practice (section 5.2.4).

We think that the effort of decentralizing a national identity system by introducing a complex network such as KERI it is not justified by the mean of the swiss e-ID, which is to provide an identity controlled and issued by a centralized entity. The feature of recovery from compromise may not be fully exploited as most of the time citizens identifiers are not critical and in practice it may be easier, or more frequent, that an identifier is simply substituted. Furthermore KERI does not stop an attack, even after a rotation an attack may continue, which does not bring any security advantage compared to other systems, PKI for example.

KERI does not provide consensus, does not give guarantees about recovery from an attack and we expect scalability issues when we want to enforce security proprieties as in the presented model. KERI may provide attack detection and probabilistic security, but this may not be sufficient for the swiss e-ID trust infrastructure.

We also see some problem on a quick implementation, as the evaluated library is not mature enough and keripy, the other library, is missing features (watchers). Of course one ad-hoc implementation could be produced, but this would require a longer development time. The time factor is important considered the introduction of the swiss e-ID by 2026.

6.2.2 Trust Registry

While the focus of this thesis was to evaluate KERI as possible base registry for the swiss e-ID, we identified an alternative, promising use-case inside the

trust registry. The trust registry is used by the state to authorize or recognize entities. An example is to define which entities are entitled to issue an e-ID, or to recognize an institution, such as an university, or to authorize a bank to make loans.

This use-case is very similar to the GLEIF use-case. In particular we do not deal with short-lived credentials with strong time requirements, as trust do no change so quickly, and we work on a very different scale. Furthermore privacy would not be an issue.

For these reasons we see KERI as a better candidate for the trust registry, rather than the base registry. Further work is needed to assess this intuition and to study some design.

6.3 Thanks and acknowledgements

I would like to thank the CYD Campus of armasuisse and in particular my supervisor Dr. Martin Burkhart for giving me the opportunity to study KERI and supporting me. I thank as well my supervisors at ETH Zurich, Dr. Kari Kostiainen and Carolin Beer for following me and providing constructive feedback.

Furthermore I would like to thank Philippe Page, Robert Mitwicki, Michal Pietrus, Edyta Pawlak and the whole Human Colossus foundation for the useful meetings and exchange of opinions. In particular I thank Edyta Pawlak whith which I strictly collaborated for the development of the model in keriox.

Finally I thank Dr. Samuel M. Smith, the creator of KERI, as well as Kent Bull, part of the KERI team, for the nice discussions and explanations. A thank goes to the KERI discord community[1] which is always ready to answer your questions.

The figures on this thesis are made using Google Drawings and the matplotlib library. We used some Large Language Model (LLM) to support the production of the code for the network infrastructure and the creation of the plots.

[1] https://discord.com/channels/1148629222647148624

Bibliography

[1] A. Tobin and D. Reed (Sovrin Foundation), *The inevitable rise of self-sovereign identity*, 2017. [Online]. Available: https://sovrin.org/wp-content/uploads/2017/06/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf.

[2] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, *Automatic Certificate Management Environment (ACME)*, RFC 8555, Mar. 2019. DOI: 10.17487/RFC8555. [Online]. Available: https://www.rfc-editor.org/info/rfc8555.

[3] Bundesamt für Justiz, *Bundesgesetz über den elektronischen identitätsnachweis und andere elektronische nachweise*, Work in Progress, 2023. [Online]. Available: https://fedlex.data.admin.ch/eli/fga/2023/2843.

[4] P.-A. Champin, G. Kellogg, and D. Longley, "JSON-ld 1.1," W3C, W3C Recommendation, Jul. 2020. [Online]. Available: https://www.w3.org/TR/2020/REC-json-ld11-20200716/.

[5] P. Dunphy, *A note on the blockchain trilemma for decentralized identity: Learning from experiments with hyperledger indy*, 2022. arXiv: 2204.05784 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2204.05784.

[6] European Banking Authority (EBA), *Pillar 3 data hub processes and possible practical implications*, 2023. [Online]. Available: https://www.eba.europa.eu/sites/default/files/2023-12/d5b13b4d-a9dc-4680-8b7c-0a3a4c694fac/Discussion%20paper%20on%20Pillar3%20data%20hub.pdf.

[7] D. Fett, K. Yasuda, and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)," Internet Engineering Task Force, Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-10, Jul. 2024, Work in Progress, 91 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/10/.

[8] GLEIF, *Verifiable lei (vLEI) ecosystem governance framework v2.0, technical requirements part 1: Keri infrastructure*, 2023. [Online]. Available: https://www.gleif.org/vlei/introducing-the-vlei-ecosystem-governance-framework/2023-12-15_vlei-egf-v2.0-technical-requirements-part-1-keri-infrastructure_v1.2_final.pdf.

[9] I. Grigg, "The ricardian contract," in *Proceedings. First IEEE International Workshop on Electronic Contracting, 2004.*, 2004, pp. 25–31. DOI: 10.1109/WEC.2004.1319505.

[10] C. Heimann. "Discussion paper: Initial technological basis for the swiss trust infrastructure." (2023), [Online]. Available: https://github.com/e-id-admin/open-source-community/blob/main/discussion-paper-tech-proposal/discussion-paper-tech-proposal.md (visited on 07/17/2024).

[11] Hyperledger White Paper Working Group, *An introduction to hyperledger*, Aug. 2018. [Online]. Available: https://www.hyperledger.org/learn/white-papers.

[12] B. Laurie, A. Langley, and E. Kasper, *Certificate Transparency*, RFC 6962, Jun. 2013. DOI: 10.17487/RFC6962. [Online]. Available: https://www.rfc-editor.org/info/rfc6962.

[13] T. Looker, V. Kalos, A. Whitehead, and M. Lodder, "The BBS Signature Scheme," Internet Engineering Task Force, Internet-Draft draft-irtf-cfrg-bbs-signatures-06, Jun. 2024, Work in Progress, 117 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/06/.

[14] N. van der Meulen, "Diginotar: Dissecting the first dutch digital disaster," *Journal of Strategic Security*, vol. 6, no. 2, pp. 46–58, 2013, ISSN: 19440464, 19440472. [Online]. Available: http://www.jstor.org/stable/26466760 (visited on 07/24/2024).

[15] P. Page, P. Knowles, and R. Mitwicki, *Distributed governance: A principal-agent approach to data governance – part 1 background & core definitions*, 2023. arXiv: 2308.07280 [cs.CY]. [Online]. Available: https://arxiv.org/abs/2308.07280.

[16] D. Reed, M. Sabadello, M. Sporny, and A. Guy, "Decentralized identifiers (DIDs) v1.0," W3C, W3C Recommendation, Jul. 2022. [Online]. Available: https://www.w3.org/TR/2022/REC-did-core-20220719/.

[17] S. M. Smith, "Key event receipt infrastructure (keri)," *arXiv preprint arXiv:1907.02143*, 2019.

[18] S. M. Smith, "Authentic Chained Data Containers (ACDC)," Internet Engineering Task Force, Internet-Draft draft-ssmith-acdc-03, Jul. 2023, Work in Progress, 102 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ssmith-acdc/03/.

[19] S. M. Smith, "Composable Event Streaming Representation (CESR)," Internet Engineering Task Force, Internet-Draft draft-ssmith-cesr-03, Jul. 2023, Work in Progress, 51 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ssmith-cesr/03/.

[20] S. M. Smith, "Out-Of-Band-Introduction (OOBI) Protocol," Internet Engineering Task Force, Internet-Draft draft-ssmith-oobi-01, Jul. 2023, Work in Progress, 22 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ssmith-oobi/01/.

[21] S. M. Smith, "Self-Addressing IDentifier (SAID)," Internet Engineering Task Force, Internet-Draft draft-ssmith-said-03, Jul. 2023, Work in Progress, 11 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ssmith-said/03/.

[22] S. M. Smith and P. S. Feairheller, "Issuance and Presentation Exchange Protocol," Internet Engineering Task Force, Internet-Draft draft-ssmith-ipex-00, Jul. 2023, Work in Progress, 16 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ssmith-ipex/00/.

[23] E. Stark *et al.*, "Does certificate transparency break the web? measuring adoption and error rate," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 211–226. DOI: 10.1109/SP.2019.00027.

[24] K. Yasuda, T. Lodderstedt, D. Chadwick, K. Nakamura, and J. Vercammen, "Openid for verifiable credentials," unpublished (Version: 2nd Editor's Draft), 2022. [Online]. Available: https://openid.net/wordpress-content/uploads/2022/06/OIDF-Whitepaper_OpenID-for-Verifiable-Credentials-V2_2022-06-23.pdf (visited on 07/17/2024).

Appendix A

Network Test Results

We report the measured timings from the experiments. The timings are expressed in ms. For display reasons, we wrote * for the column representing the tally value of the column to the right.

Baseline

id	users	success	*	wit	*	wat	test 1	test 3	test 4
0813-10:54	500	500	3	4	3	4	5648	598	1893
0813-11:35	1000	1000	3	4	3	4	5324	545	1814
0813-11:42	2500	2500	3	4	3	4	5334	502	2041
0813-16:46	4000	3990	3	4	3	4	6295	513	3430
0813-12:53	5000	4990	3	4	3	4	7835	486	7061
0814-09:31	7500	7440	3	4	3	4	15766	496	26151
0814-10:29	10030	9770	3	4	3	4	37721	480	51115
0813-14:20	500	510	3	4	3	4	5603	538	1896
0813-14:25	1000	1010	3	4	3	4	5650	545	1912
0813-14:31	2500	2510	3	4	3	4	5581	527	2030
0813-16:58	4000	4000	3	4	3	4	6360	489	3347
0813-14:45	5000	4990	3	4	3	4	7559	497	6925
0814-10:02	7500	7460	3	4	3	4	18737	492	25390
0814-11:06	10030	9650	3	4	3	4	38755	488	51079

More Witnesses

id	users	success	*	wit	*	wat	test 1	test 3	test 4
0813-12:08	1000	1000	5	7	3	4	9560	834	4820
0813-12:15	1000	1000	6	8	3	4	12659	816	6718
0813-12:21	1000	1000	7	10	3	4	14070	886	8608
0813-13:33	1000	980	8	12	3	4	17320	1067	11758
0813-15:40	1000	1010	5	7	3	4	9765	672	4705
0813-15:46	1000	1000	6	8	3	4	11656	801	6597
0813-15:56	1000	1010	7	10	3	4	15147	924	8970
0813-16:02	1000	940	8	12	3	4	18270	1359	11446

More Watchers

id	users	success	*	wit	*	wat	test 1	test 3	test 4
0813-13:48	1000	1010	3	4	5	7	6968	910	1845
0813-13:54	1000	1010	3	4	6	8	7885	1143	1868
0813-14:00	1000	1010	3	4	7	10	8700	1327	1851
0813-14:07	1000	1000	3	4	8	12	9921	1593	1926
0813-16:16	1000	1010	3	4	5	7	7275	905	1961
0813-16:22	1000	1010	3	4	6	8	8094	1100	1927
0813-16:28	1000	1010	3	4	7	10	8933	1309	1915
0813-16:35	1000	1000	3	4	8	12	9805	1682	1941