Open Key Exchange

Introduction

openKex Foundation

Version 0.4, 2017-06-12

Table of Contents

1. About this document

This document will give a brief introduction to Open Key Exchange (abbreviated as "openKex").

More detailed documents will follow covering full requirements (roles, use cases and domain rules), technical design (protocols, stability and scalability) and security (threat scenarios).

1.1. Status

This document is currently in draft status. It represents the prototype phase of the project.

1.2. History of changes

Version	Date	Description
0.1	2017-01-10	Initial Draft
0.2	2017-01-20	Fixed Typos, Clarifications, Added Chapter 5
0.3	2017-04-12	Added Chapter "Solution Aspects"
0.4	2017-06-12	Initial Public Version

2. Overview

openKex is a new approach to manage digital identities.

2.1. Motivation

Digital identities already exists in different flavors, we all use them in our daily life. Common examples are your email address your phone number or social media accounts.

This chapter will show the motivation to create openKex by looking at the shortcomings of current solutions.

2.1.1. Ownership

Whenever you enter your credentials in a web browser you are using a remote account, you don't own the according identity - the service vendor does.

The vendor may claim not to store your cleartext password but he can obviously read it during transmission.

2.1.2. Control and Privacy

Not owning your identity has serious consequences, you loose the option to choose what happens with your personal data.

Of course service vendors have privacy statements but they also force you to sign off "shady" terms that fill dozens of pages and will be hard to understand without the help of your lawyer. Accepting these terms without reading or questioning the details is considered as normal ("Working as designed").

Even if the service vendor claims to protect your privacy cases of abuse, unauthorized access or lawful interception can happen. From the technical point of view the vendor can expose or sell your data, impersonate you, modify your account or even delete any data.

Creating profile information by tracing your activity to make you an attractive target for commercials is already best practice, the terms you have not read cover these kind of things.

2.1.3. Vendor Lock-In

Vendors realized that identities are good business. They offer "free" services to attract users. They provide easy ways of integration and migration - it takes a "single click" in your web browser.

For the gain of comfort users accept to hand over their data to a single vendor. The dangers of allowing such a monopoly situation are obvious.

2.1.4. Trust and Transparency

The only relevant point where trust can be (theoretically) formally validated is when users connect to a web site via https. The connection shall authenticate the site owner via X.509 certificates.

This validation is based on a list of more than one hundred authorities that are more or less "hardcoded" in the web browser, but this list is never presented to the user. The user is expected to

"implicitly" trust each of these companies (most of which he never heard about). Instead of confronting the user with these confusing details the browser shows a small green icon to indicate "security".

2.1.5. Complexity

Complexity is a general problem that escalates as solutions evolve, as example we continue with the previous question of trust.

Assume that an ambitious user would like to clarify the confusing trust relations. To review the list of trusted authorities he will need to click through an endless sequence of popup dialogs each showing a long list of properties. But these properties cannot be interpreted without specifications, so the next hypothetical step would be to collect all relevant specifications. Reading and understanding all related RFCs is a hard "expert's job" but there is still more to do. Each authority publishes documents called "Certificate Practice Statement" and "Certificate Policy" adding a few hundred more documents to read, each with approximately 50 pages. This is the point where the hypothetical ambitious user finally gives up.

Another interesting example for complexity might be the web browser (displaying the little pop up dialogs in the previous case). Today's web browsers have roughly 15 million lines of source code, try to imagine the job of verifying that the considered browser is free of bugs.

2.1.6. Centralized and Closed Solutions

Previous chapters already touched this topic but we like to emphasize it again. Vendors create proprietary solution that store all data in their central data centers. Some deliver their own closed identity system that is required to use their service offer.

Also systems that only deal with authorization consider itself as only valid source of identities or exclusively define the list of trustworthy authorities.

2.1.7. Other Problems

The actual list of issues is longer but we skip the rest with the conclusion that we did not find a solution that could be easily adapted to manage identities in the way we prefer.

Even those rare solutions which take care about privacy turned out to rely on their own "centralized" data and their own identity logic that creates yet another closed user group.

2.2. High Level Requirements

Based on the problems listed in the previous chapter we define the following user requirements for the openKex solution. They define fundamental rules for the solution design.

- Enable reliable and endurable ownership of your digital identity
- Allow to control what personal data will be exposed
- Rely on a minimum of well defined trust relations
- Avoid obligated central authorities

Moreover we add the following points for implementation.

- Create an open source solution
- Focus on simplicity and transparency
- Offer open interfaces for flexible integration
- Offer the service for free (at least for personal use)
- Minimize involved data to improve privacy (in German "Datensparsamkeit")
- Offer different trust models as option (freedom of choice)
- Allow federated operation

2.3. Scope

openKex tries to take care of identities without adding more aspects, it might appear abstract without further usage scenarios. These usage scenarios will hopefully grow over the time with the number of integrated interfaces or systems.

The separation gives us the options to keep a clear focus on core topics, setting identities free by creating a simple and open solution available for everyone.

3. Solution Approach

This chapter will describe some basic design assumptions we made to fulfill the requirements of openKex.

3.1. The openKex Identity

3.1.1. Identifier

We need a globally unique and endurable identifier. There shall be enough identifiers available. There shall be no special or rare identifiers (they would cause avidity and shortage).

The technical representation will be a random number with 48 bits allowing 10^{14} variants. We call it `KexID`. A hex string representation looks like this example `8F354DA3F642`. It fits in a small QR code (25x25).

3.1.2. Key Holder

In the world of digital communication the best way to prove your identity is *to hold a key*. More precisely we are talking about a key based on an asymmetric cryptographic algorithms like RSA or ECC. Digital Signature algorithms can prove that certain information was exclusively confirmed by the owner ("holder") of the private key while the according public key can be distributed freely.

The user proves to be the correct key holder with the following methods (similar to authentication "factors"):

- Ownership: user owns (holds) the key
- Knowledge: user knows the password or pin that is used to protect the key
- Inherence: user presents something (e.g. a fingerprint) that is used to protect the key

The first factor is required. At least one of the others are recommended (note that the password we are taking about here is *not* sent to someone else's server and it is always a second or third factor).

As long as the owner properly protects the private key it will be a good prove of identity.
The openKex user will hold keys that prove ownership of the `KexID`.

3.1.3. Lifecycle

Key pairs will not last forever. Algorithms may become irrelevant or "weak" over time, keys may get lost or compromised, using keys less frequently improves certain security aspects.

For this reason we do not use public keys as identifier (another reason is that public keys may be significantly larger). This approach preserves the chance to keep using the same identity in specific problem cases.

As a consequence the openKex registry is required to enable the user to "claim" his `KexID`. Claiming the `KexID` requires an initial key pair, this key pair shall be replaced rather frequently (a few months) to extend the claim (an option might be to prolong key validity). The short key lifecycle will increase security and avoid stale identities, obviously comfort will be reduced.

3.1.4. Secure Elements

Secure Elements help to protect private keys with specialized hardware. Key pair generation and signing happens locally on the device protecting the private key from being exported or stolen by different measures. In a simple case this operation will require a password, multiple failed password attempts will permanently lock the device.

Specific integrated circuits are available for less than one dollar. Actually we are surrounded by secure elements - your mobile phone, your credit card and your laptop already contain such elements. Unfortunately some of these elements are used in a centralized and restrictive manner against the user's interests. They can be used by vendors to enforce what software runs on a specific device, the so called "owner" who paid for the device has to accept and trust the vendor defined closed source software.

We will try to promote and integrate *free secure elements* that can be used to protect your identity.

3.2. The openKex Registry

The openKex Registry shall act as public and global storage of identities. Any data stored in the registry shall be signed by the user (and some other contributors). Any data is appended to the registry to keep the history of changes transparent, no data can be deleted.

The openKex registry shall be operated by a federation of contributors. These contributors will ensure that the registry is not compromised. Consistency is based on majority decisions.

The user who initially claimed his `KexID` can append information to his identity. He might want to add personal information, references to other IDs like e-mail addresses, etc. The registry shall check all data based on well defined rules. Certain information can be validates by contributors (e.g. by checking e-mail via confirmation links and testing for uniqueness). The actual range of use cases is not yet settled.

3.3. A First Summary

Let's reconsider what assumption we made and match them against the high level requirements.

So far we did not assume much, this will contribute to *simplicity*.

The concept of the key holder will ensure *ownership*, it can be improved by using secure elements.

The registry rules will ensure *control*, only the user can add information to his identity.

The *trust* required by the user applies to the registry. The open implementation and the well defined rules for federated operation will hopefully create enough confidence.

4. Solution Details

Solution details shall give a first insight of what we planned and achieved in the prototype phase. Note that information in this chapter is work in progress and might contain flaws and errors. There are missing bits so the picture may appear incomplete.

4.1. System Components

We first look at the sketch of potential components of the system (openKex is abbreviated to KEX). User X represents one out of millions of users. Operator A represents one out of thousands of operators, four servers are shown to expose the consensus interface. The Key Store API defines operations to generate keys and create signatures. As example the picture shows a smart card that is used as secure element.

The diagram uses UML symbols while not following a specific diagram type. Boxes are "things", dotted lines indicate interaction, full lines represent technical interfaces.

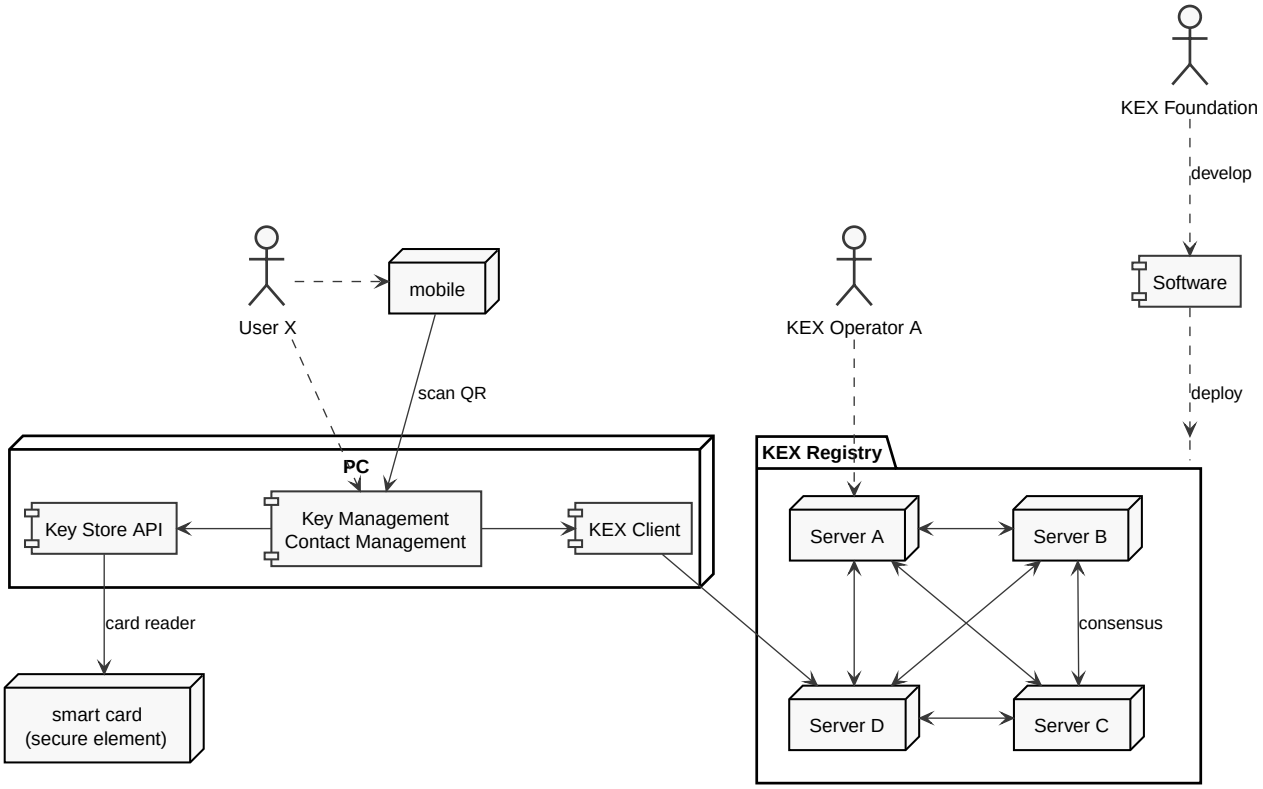

Figure 1. System Components

4.2. Roles and Responsibilities

4.2.1. openKex Foundation

Shall be a non-profit organization.
Defines rules for openKex and provides according specifications and (reference) software.
Is responsible to organize operations.

4.2.2. Registry Operator

Operates (one or more) registry servers.
Follows foundation rules (might be member of the foundation).

4.2.3. User

A natural person (maybe also an organization) using openKex.
Owns an identity referenced by a `KexID`.

4.2.4. Other Roles

In future more roles will be needed for specific use cases (e.g. Validator or Identity Provider).

4.3. Statements and Rules

Statements are the primary data elements stored in the registry.

- Each statement is signed by an issuer.
- The issuer is identified by a `KexID`.
- For each type of statement specific fields and rules are defined.
- Valid statements will be added to the registry.
- The registry confirms publishing date (the statement was created before this date).

The following table shows examples of statement types:

Table 1. Statement Types

Type	Issuer	Statement (textual)	Fields
Claim	User	I want to claim `KexID1` with public key `P1`	`P1`
Add Attribute	User	I add attribute of name `N1` with value `V1` to my identity	`N1`, `V1`
Add Masked Attribute	User	I add attribute of name `N1` with masked value `MV1` to my identity	`N1`, `MV1`
Certificate Owner	User	I own certificate type `T1` and id `C1`, signature `S1` proves ownership	`T1`, `C1`, `S1`

The rules for type Claim are for example:

- `KexID` is not yet claimed
- Public key `P1` matches the signature

The complete and consistent list of statements and rules is under construction.
It it one critical key element of the openKex solution.

4.3.1. Statement Details

For statements we need a signature algorithm defined like this:

- Variables: `M1` message, key pair: `PrK1` private key, `PuK1` public key

- Signature SiM1K1 (bytes): `SiM1K1 = sign(M1, PrK1)`
- Verification V1 (boolean, true is OK): `V1 = verify(M1, SiM1K1, PuK1)`

4.3.2. Statement Example: Certificate Owner

For the claim *"I own certificate C1"* the user needs to prove

- that he exclusively owns `KexID1` registered with `PrK1` (private), `PuK1` (public)
- AND that he exclusively owns the considered certificate `C1` with `PrC1` (private), `PuC1` (public)

Steps to create statement

- A: `SiC1K1 = sign(PuC1, PrK1)`
- B: `SiK1C1 = sign(PuK1, PrC1)`
- C: publish `SiC1K1, SiK1C1, PuC1` (`PuK1` is known as key of publisher)

Note: `PuC1` in A has to match `PrC1` in B and `PuK1` in A has to match `PrK1` in B - this "links" both signatures

Steps to verify statement (`PuC1` and `PuK1` used twice - this validates the "link")

- A: `V1 = verify(PuC1, SiC1K1, PuK1)`
- B: `V2 = verify(PuK1, SiK1C1, PuC1)`
- C: if V1 and V2 is true the claim is proven

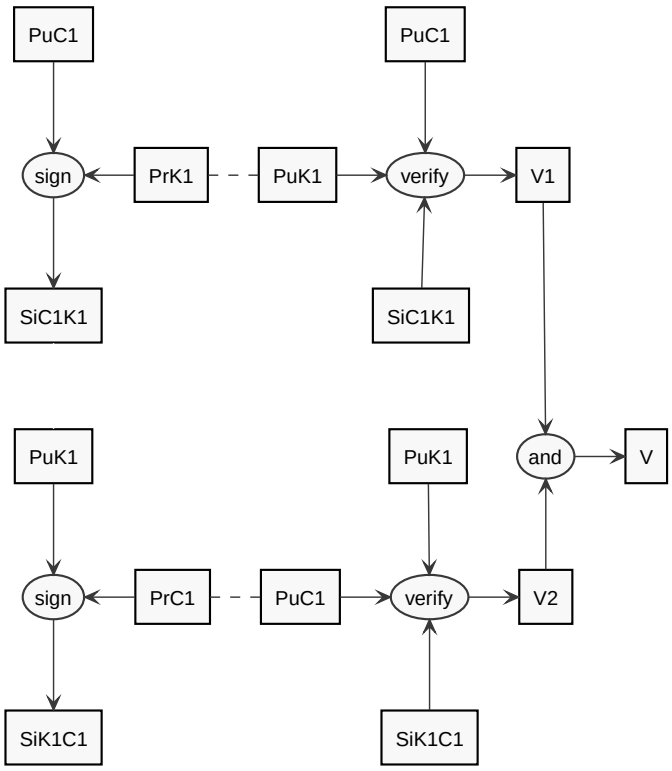

Figure 2. Creation (left) and Validation (right) of Statement 'Certificate Owner'

4.4. Protocols

4.4.1. Client Protocol

The client can talk to any server, talking to a local server is preferred. The client will be authenticated once the initial ID was claimed. The protocol will add measures for rate limiting (e.g. client challenge with "proof of work").

This sequence shows an initial claim conversation.

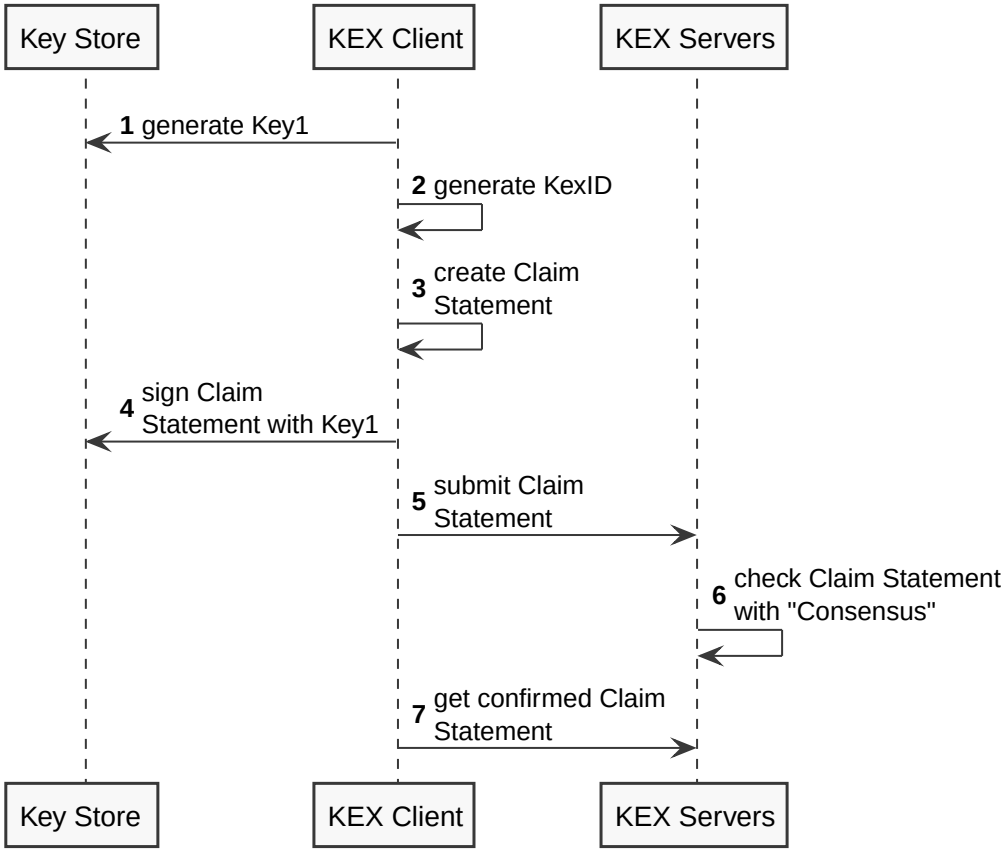

Figure 3. Client Protocol

1. Generate key pair via key store API [1]
2. Generate KexID as 48 bit random
3. Create Claim statement
4. Sign statement via key store API [1]
5. Send statement to server
6. The rules will be checked by all servers, see next chapter for "Consensus"
7. After next round of "Consensus" the confirmed statement will be available if checks were OK.

[1] in case of a secure element authentication will be required (e.g. pin entry, fingerprint scan)

4.4.2. Server Protocol (Consensus)

This sequence diagram shows a simple round based consensus protocol. Servers talk to each other to exchange all new client statements (phase "Fetch"). When a round succeeds with consensus all servers will hold the identical new set of statements.

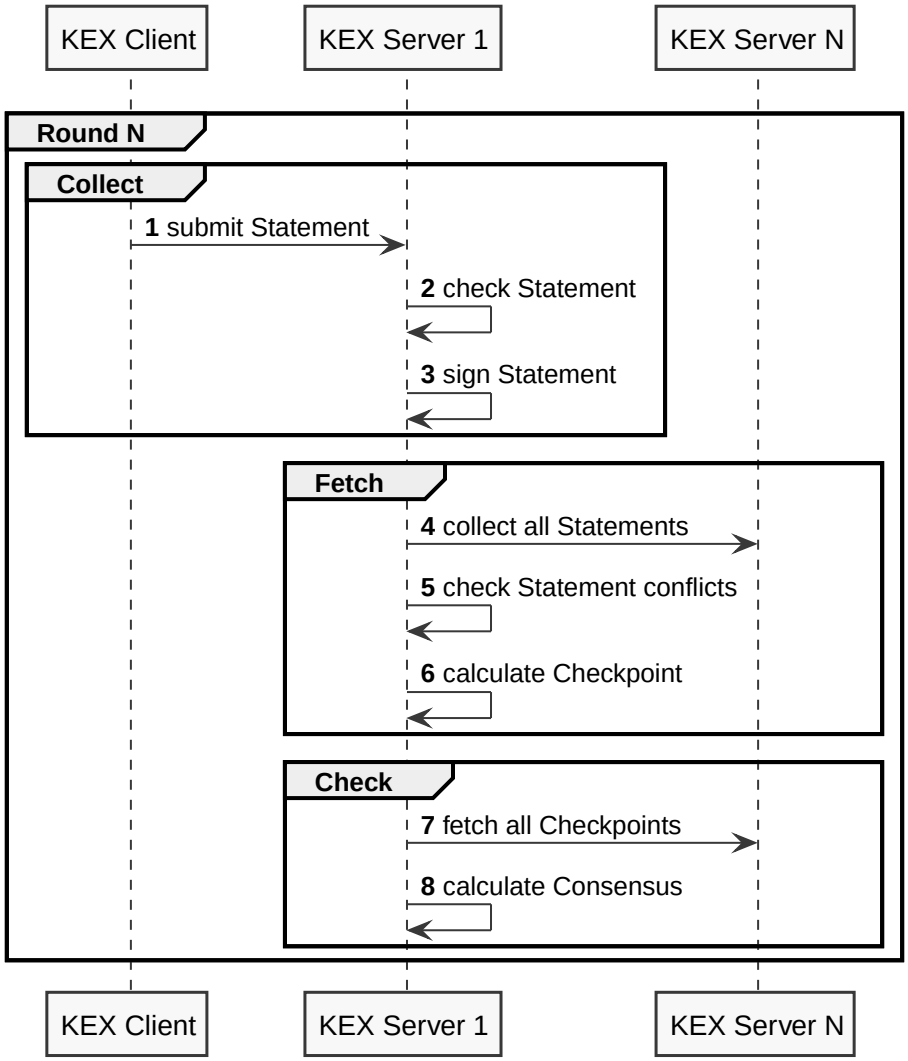

Figure 4. Server Protocol

The consensus protocol works in phases (these phases might also be "pipelined").

In phase *Collect* users can submit their statements (1) using the client protocol. The according server will check the rules that apply to the statement (2) and sign it in case of success (3).

In phase *Fetch* servers will fetch the statements collected by the other servers (4). Certain statements can create conflicts (e.g. claims with identical `KexID`) and will be removed in this case (5). Based on all valid statements a new Checkpoint is calculated (6).

In phase *Check* each sever will fetch Checkpoints calculated by the other servers (7). If the significant majority of Checkpoints are identical the new statements are accepted (8).

4.5. Secure Element Requirements

We would like to use a secure element with the following properties:

- Hardware protected private keys (standards: Common Criteria EAL, FIPS 140-2)
- Flexible usage of key generation and signatures (no "central control")
- ECDSA (ECC 256+ Certicom) support, 10+ keys
- EdDSA/RFC 8032 (as preferred Certicom alternative)
- Flexible interfaces to support PC and mobile phone (see [1])
- Open hardware, open software
- Cheap

[1]: usual card interfaces are wired (ISO 7816) and NFC (ISO 14443). both have issues:
Wired: requires a reader, is hard (and awkward) for mobile phone.
NFC: might work with mobile phones but has issues, a reader is required for the PC.

We would prefer Bluetooth as interface but there are no standards and no devices available.

The current prototype supports Smartcard HSM. It fulfills the first three requirements and is relatively cheap but requires a reader. We use ECDSA with `secp256k1`.

5. Solution Aspects

We will now try to investigate how the solution sketched in chapter Solution Details serves the required purpose and what further steps are required.

5.1. Trust

From the user's perspective there are different parties he needs to trust.

5.1.1. Trust the openKex Foundation

The openKex Foundation is responsible to set up the solution correctly. The quality and clarity of documentation and code will (at a certain project phase) enable qualified reviews that justify this trust.

5.1.2. Trust the openKex Operator

The operator needs to follow rules defined by the foundation. There are two major aspects:

Enable reliable and transparent operation of the registry. This implies running valid software, setting up and protecting keys correctly and ensuring availability. (Note that by design a single operator will not be able to circumvent transparency.)

Protecting user's privacy by not recording connection data (IP addresses, data exchanged via API calls, etc.).

The second aspect cannot be validated, but as all operators are equivalent the user may choose one he specifically trusts.

5.1.3. Trusted Secure Element

It might be useful to prove that a key is actually hardware protected. This requires an additional signature with a trusted vendor key (or key chain). The vendor needs to prove that correct production procedures and correct device design proves the claim (The prototype implements such a validation for Smartcard HSM).

Note that this device key has no relation with and imposes no limit to the actual key usage.

Note that such a vendor signature could expose an identifier that reveal users identity via the supply chain. Such a link shall be avoided via logistical or technical measures.

5.2. Privacy

Assuming that the operator properly protects connection data there are no obvious privacy issues as any information in the registry is public. The user needs to understand this and act accordingly (e.g. publish only necessary data, use multiple `KexIDs` for different purposes, etc.)

What the platform cannot solve is the possibility to link personal information with user's `KexID`. There is an industry that makes money out of collecting and linking personally identifiable information. In the hypothetical case when the `KexID` becomes relevant it will be yet another collected identifier. Circumventing these effect is (obviously) beyond the current project scope.

5.3. Security

Security is based on three assumptions:

- Relevant private keys (i.e. foundation and operator keys) are well protected.
- Statements and Rules are complete and correct to fulfill the required features.
- Server Protocol (Consensus) ensures that valid data will be distributed transparently (i.e. not forgeable, not removable)

Each of these assumptions will be analysed in detail in the document "Open Key Exchange - Security".

5.4. Abuse

Creating an open platform will enable a certain degree of abuse. We currently try to address such cases with different approaches.

- Making rules as strict as possible (considering abuse scenarios).
- Slowing down certain client operations (e.g. by requesting "proof of work").
- Allow operational measures (without relevant reduction of privacy protection).
- One "drastic" option would be to limit usage to hardware based keys only, this could make sense when cheap devices are available.

6. Related Research and Solutions

The previous chapters tried to describe the shortest path from the requirements to the solution sketch.

Here we try to give a brief overview of current research and existing solutions. If appropriate we will clarify where we found inspiration and what we deliberately excluded for our solution approach.

6.1. Research

A good starting point for research papers is "SoK: Secure Messaging" [SoK] with 137 references. The openKex focus matches chapter "IV. Trust Establishment".

The site "Rebooting the Web of Trust" [RWOT] tries to refresh the PGP based logic called "Web of Trust" [PGP]. We appreciate and share their vision of *decentralized self-sovereign identities*.

6.2. Solutions

No established solution comes close to our requirements, but there are others that address similar requirements.

6.2.1. Legacy

When searching for a format that would represent what we call "Statement" we decided to skip both existing legacy formats, as they do not serve our need for simplicity and clarity.

- X.509 [X509] is far to complicated, specific topics are delegated to the authorities, it was made for "centralized" organizations (as single point of authority).
- PGP [PGP] is not flexible enough and suffers from unclear semantics and bad usability. The idea "Web of Trust" did not work out properly.

6.2.2. Keybase

Keybase [KB] is a trust establishment scheme allowing users to find public keys associated with social network accounts (see [SoK]).

What we might imitate is the "linking" of public accounts via publishing of specific confirmation messages (leaving the actual significance of such accounts to the judgement of the user).

What we did not like is the closed source (assumed "centralized") server, the command line interface and the heavy dependency on GNU Privacy Guard (GPG). The account administration is based on a password.

6.2.3. Certificate Transparency

Certificate Transparency [CT] implements a Transparency Log as defined in [SoK]. It tries to "un-break" some X.509 authority problems but is not meant to be used by individuals.

Appendix A: References

[SoK] SoK: Secure Messaging, 2015 http://cacr.uwaterloo.ca/techreports/2015/cacr2015-02.pdf

[RWOT] Rebooting the Web of Trust http://www.weboftrust.info

[PGP] The PGP Paradigm, 2015 https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust/raw/master/topics-and-advance-readings/PGP-Paradigm.pdf

[KB] Keybase Docs https://keybase.io/docs

[BS] Blockstack Papers https://blockstack.org/papers

[CT] RFC 6962, 2013, Experimental https://tools.ietf.org/html/rfc6962

[X509] https://en.wikipedia.org/wiki/X.509, RFC 5280 (with long list of related RFCs)