P. Kravchenko B. Skriabin O. Kurbatov

Engineer's Guide to

Financial Internet

Distributed Lab

DISTRIBUTED LAB

P. Kravchenko, B. Skriabin, O. Kurbatov

ENGINEER'S GUIDE TO FINANCIAL INTERNET

Authors' Test Edition

Kharkiv
2019

UDC 004.738.5:336]:007-057.21](07)=111
K78

Authors:

P. Kravchenko, B. Skriabin, O. Kurbatov

Kravchenko P.

K78 ENGINEER'S GUIDE TO FINANCIAL INTERNET. Authors' test edition / P. Kravchenko, B. Skriabin, O. Kurbatov. – Kharkiv: PROMART, 2019. – 224 p.: 96 figures, 5 tables.

ISBN 978-617-7634-52-1

The book describes the concept and design of Financial Internet. In the first part, the concept is being defined conceptually and the corresponding context is given: limitations of traditional systems, main principles and components of Financial Internet, etc. Next, the book describes the architectural features of an accounting system, proposes an approach for identifying users in a decentralized environment, describes the issues of interoperability, main principles of the data storage, key management, and more.

The book is intended primarily for engineers/architects that develop financial applications, and its aim is to provide a stack of technologies behind Financial Internet and to create uniform standards of how accounting systems should be built.

UDC 004.738.5:336]:007-057.21](07)=111

ISBN 978-617-7634-52-1

Devoted to future generations…

TABLE OF CONTENTS

INTRODUCTION

Why is sending money not as easy as sending an email? Why are there so many different payment systems available that do not integrate well with each other? Why is paying on the internet so inconvenient?

Some merchants accept only internationally recognized credit cards such as Visa and Mastercard. Other merchants accept only locally-issued credit cards. In contrast, vending machines usually accept only small bills or coins. And, the minimum amount of a credit card transaction can be 10 euro. Furthermore, it is at least difficult and often impossible to send money from a bank account to a credit card. Even worse, consider land registries, warehouse receipts, and intellectual property; in the worst cases, they are still accounted for on paper, and even if there is a digital system in place, it usually does not provide an open API and thus an easy customer journey.

What's interesting is that in many cases, people don't even notice that something isn't right. We got used to the world we are living in. We can't even imagine that it could be different. The fact, however, remains that sending a payment will at some point be as simple as sending an email or a message on social media. Think about it, would a person living in the 19th century have ever imagined that people will be able to communicate with each other easily and instantly while being located in different parts of the world? While today, we take it for granted. Beyond that, when was the last time you wrote a letter with a pen out of necessity?

This book tells the story about Financial Internet in the way we see it. Financial Internet is built for consumers. It is needed for financial institutions, while it will bring a substantial change in the role of banks and their services. It will replace the cumbersome process of managing one's assets with a few taps on a smartphone. Financial Internet is expected to represent ownership rights of assets in a unified way and to provide access to accounting systems using the uniform protocols (such as HTTP).

WHO THIS BOOK IS FOR

In this book, we will describe Financial Internet as a technology stack. We will cover its main components, processes, four main entities—private data, identity, transactions, and assets—their lifecycle as well as the related considerations, namely security, privacy, and interoperability.

Therefore, the book is best for:

Engineers/architects that develop financial applications: digital payment/remittance systems, wallets, asset management systems, crowdfunding systems, etc.

Members of technical committees who are busy with unification and standardization of protocols and approaches.

Students who want to learn how financial applications work.

Innovation managers/engineers who want to see how new technologies can benefit their businesses.

ABOUT DISTRIBUTED LAB

Distributed Lab (https://distributedlab.com/) was founded in 2014 as an R&D company. We started to work on a wide variety of projects from wallets to digital banks, all operating in the field of accounting and management of assets. It became obvious that dozens of thousands of engineers were reinventing the wheel again and again building very similar systems.

We saw the potential that the advancements in cryptography and blockchain approaches could bring to business operations and became extremely excited about it. This is how our mission, to build the Financial Internet, was formed in 2015.

Our mission may sound too ambitious, but we believe that it can be achieved step by step. The closest goal is digitization (tokenization) of asset management systems. To make this happen we have created a framework called TokenD (https://tokend.io/)—therefore since 2018 we became entirely a product company. We see our product as a Wordpress for digitization. It is needed to build your own system faster, cheaper and less risky than ever before. It appeared as a consequence of years of work on different client projects that as we then realized,they were essentially asking for the same thing. This "thing" is a full stack reference accounting system which is secure, has wallets for users, a ledger for assets, an internal payment system, gateways, an exchange, modules that manage identities and roles, the lifecycle of assets, and so on. It doesn't care which ownership rights it deals with—money, securities, commodities or real estate deeds. Having combined everything mentioned, TokenD allows one to launch a digital asset ecosystem within weeks for a fraction of the cost of the in-house development.

During the crazy hype period of 2017, we stayed calm and didn't participate in any ICO while being focused on the search for real business value that blockchain and related technologies can deliver. We commit ourselves to the long journey of transforming the world with technology.

TERMINOLOGY AND COMMON SENSE

Since we've been working on the edge/mix of traditional fintech and blockchain, we need to clarify certain terminology used in this book as well as to give a brief overview on the specific angle from which we envision the well-known concepts.

For traditional CTOs / architects / full stack engineers / BAs / PMs:

Transaction

A cryptographically signed set of operations that initiates a change in the state of some ledger.

Ledger

A database that stores information about accounts, their balances, and permissions.

Node

A hardware and software system that stores the full history of transactions and makes decisions to update the ledger state.

Distributed ledger

A ledger that is managed and stored by multiple parties.

Blockchain

We won't be using the word "blockchain" (for the most part). For an engineer, "blockchain" (or DLT) should mean *a secure method of storage and synchronization of a ledger between independent parties who do not necessarily trust each other*. This method is based on 1) cryptographic mechanisms (hashes, digital signatures) and 2) consensus protocols. Their application will be described in this book with no cliches such as "blockchain will save the world" or alternatively "blockchain is slow and not a magic pill", but rather it will be derived primarily from the security model requirements for accounting systems. And obviously, techniques in this book are NOT about issuing cryptocurrencies and getting rich overnight.

Why are we talking about DLT/blockchain in the first place? After all, this may seem strange in the context of applying it to a centralized business. In fact, it is not, and there are several reasons for that. They stem from the requirements

to IT systems that should provide an adequate level of security while operating on the Internet:

1. Integrity of transactions and their history should be controlled via a digital signature.
2. Non-repudiation of every action in the system should be guaranteed.
3. Every user/admin should have a digital identity.
4. The IT infrastructure should have no single point of failure (this applies even to the case of a few departments of one business or multiple branches of one bank).
5. Decision-making in a distributed environment (stakeholders of a business are distributed) should be protected by cryptography and the consensus mechanism.
6. Reconciliation between different businesses should be based on mathematics and not on their trust towards each other or an intermediary.

If one tries to implement such requirements from scratch, he/she would end up with the techniques described in this book.

For blockchain engineers:

Bitcoin

The first permissionless, censorship-resistant financial system. It has already found its niche, and it is challenging the belief that money should always be tied to a state (as it was with religion, information production, or the freedom of speech). But this isn't the focus of this book. We are trying to show how the existing approaches some of which were introduced first in Bitcoin can be applied to digitize ALL assets—whether they are issued by a startup in the form of a share, by a state for fiat currency, by a warehouse for warehouse receipts, etc. So we will mostly discuss the situation where identified parties want to transact with each other.

Utility token issuance

Our ideas are far from that. This was popular during the ICO hype period in 2017 when people blindly copied Bitcoin approach and tried to tie it to a centralized business (which makes no sense).

Token

To avoid confusion, we won't be using the word *token* at all. The reason for this is that most people use words *token*, *coin*, *cryptocurrency*, *digitized/tokenized asset* interchangeably while meaning completely different things. The closest definition for a *token* is *an accounting unit used to represent the user's balance of some asset* (it could be a currency, commodity, security, etc.). Token is NEVER an asset—it is just the representation of ownership rights for an asset. Therefore we won't be using terms such as *"transfer of a token"* because in the end, this is all about assets, and assets don't care how and where they are accounted for—whether it is on paper, Excel, MySQL, or some distributed ledger. Instead, we will be using terms such as "*transfer of an asset*", *"transfer of a tokenized asset"*, "*transfer of an ownership right*".

Blockchain

We won't be referencing any particular blockchain if not stated otherwise. The meaning of *blockchain is a secure method of storage and synchronization of ledger between parties who do not trust each other*. The level of trust (or mistrust) between parties defines the consensus algorithm that should be used, and it can vary, from PoW and PoS to BFT and FBA, etc. We will explicitly mention public permissionless blockchain systems if we need to provide such an example.

DLT (Distributed ledger technology)

In this book, the two terms, blockchain and DLT, are identical because they both refer to the stack of technologies applied to synchronize time-ordered data between several parties.

Cryptocurrency

A permissionless accounting system with all processes decentralized: asset issuance, transaction confirmation, data storage, accounting system audit, and governance (decision-making about the updates). Anyone can become a

member of the system, hold assets, and perform transactions under the rules which are common for all.

Tokenization

The process of asset accounting and management transformation wherein the ownership of an asset is controlled directly, via a cryptographic key. It doesn't change the principles of accounting or legal compliance—we still have ledgers, accounts, balances, and transactions. But now any change in the system is always a separate transaction signed via a digital signature of its initiator.

Tokenized asset

An ownership right for an asset that is managed via a cryptographic key.

1 FINANCIAL INTERNET PRINCIPLES

Financial Internet is a situation when all accounting systems use uniform API to manage ownership rights

1.1 Limitations of existing financial accounting systems

Digitizing accounting systems is a quite recent trend but traces its roots back to the 1970s. Internet attacks were not a concern those days, which is one of the reasons why appropriate security mechanisms were not designed for the original architecture. The communication processes were not digitized until the 1990s. Regulation (reporting, AML, KYC, etc.) was lagging further—and even today it is still paper-based in many cases. As a consequence, accounting systems have inherited many legacy mechanisms, resulting in a variety of challenges.

Up to a certain point, engineers and system architects managed to cope with these issues—mostly through extending or trying to adapt the existing (legacy) architecture. However, the architecture was not designed to deal with new threats and problems.

Meanwhile, the Internet is expanding and the majority of global social processes are digitalizing. Not only our social relationships have already changed (with the appearance of Facebook and other social networks), but also the general way we interact has acquired a different, more accessible and global, nature. The next step is the transition of our financial relationships to the Internet. But, technologically we are not (yet) ready.

Eventually, we ended up in a situation where inherent problems in accounting systems became so urgent that it became apparent that the original architecture needed to be changed, if not rebuilt from scratch.

Before digging deeper, let's take a look at the situation from a bird's eye perspective and consider the most critical issues.

- *Financial accounting systems "speak different languages"*
- *Security model is no longer effective*
- *Each system identifies its users independently*
- *Processes within the system are non-transparent (sometimes even for the system owner)*
- *Digital data is vulnerable*

Financial accounting systems "speak different languages"

Even if APIs are open, they are different for each accounting system

Given the absence of open APIs, today's financial accounting systems cannot communicate with each other seamlessly (think of it as a language that you cannot learn and hence always require a translator).

Achieving at least minimal compatibility between systems is expensive in terms of time and money for both organizations and clients. Organizations need to maintain the systems and eventually end-clients pay up to $50 (sometimes even more) to conduct international money transfers.

Traditional security model is no longer effective

Even if APIs have the same specification, they do not use public key cryptography to protect messages during interaction

Existing accounting systems no longer suit the current Internet threat model [1]. They all have a common property: they were designed to protect sensitive information from *external* threats. This led to the concept of a *security perimeter*. It assumes that in order to protect the system from threats, a "perimeter" is built: firewalls, thick walls, high fences, security wires, etc. Believe or not, this is one of the reasons why we hear about devastating security breaches more and more often.

It is no longer effective to secure sensitive data by locking it behind a "strong door". One example of this is the new BYOD (Bring Your Own Device) policy, which is being rapidly adopted by companies. It means that employees are no longer supplied with corporate devices/computers but rather can use their own devices for work. While BYOD entails a great number of advantages—higher overall productivity of employees, which is the result of increased satisfaction and flexibility of work, reduced costs for a company in providing suitable resources for employees, etc.—a new problem is presented, and it primarily relates to the security concerns. Companies have to change their security policies in accordance with the new threats and risks. For example, an employee's device (storing sensitive company's data) could be connected to an

insecure network or that an employee could lose the device and untrusted parties could retrieve any unsecured data on it, etc.

Now, look at Bitcoin. It is an accounting system that maintains the record of *who owns how much* while not having any physical perimeter protection such as fences, security guards, etc. You can't just modify its ledger without having the keys from corresponding accounts.

EXAMPLE

Consider why people had stopped building forts. Forts had provided good protection against arrows, but once the gunpowder artillery was invented, forts became immovable targets, and military doctrines had to accept that (the "transition" wasn't fast, but this at least became a trigger for militarists to accept the fact that they were doing something wrong).

(Thanks to Pindar Wong for the vivid example.)

The Financial Internet's new security paradigm implies the usage of cryptographic keys. These keys are used to sign each request and thereby achieve integrity and authenticity of data [2].

Each system identifies its users independently

By and large, existing accounting systems do not support a unified digital identity. Today almost every system carries out its own identification procedure (very often with little focus on security), resulting in users undergoing the cumbersome identification and registration procedures over and over again. Hence, people have different IDs and use different authentication methods in almost every system. This leads to critical compatibility issues, a hassle for end users and additional costs for the system owners to enable compatibility somehow. It would be significantly more convenient for each user to have her own unique digital identity that works uniformly with all accounting systems.

Processes within the system are non-transparent

Initially, the closeness of software was a common practice and, moreover, was viewed as a good practice in terms of achieving security. From the

architectural perspective, trust to a designated party wasn't an issue since the mentality of the business behavior was always based on trust which was preserved by a legal and law enforcement branch of a state.

The transaction processing in traditional systems is mostly centralized, which entails a variety of problems such as the non-transparency of processes and low security against attacks on a central server as well as service denials. Further, existing solutions require a high level of users' trust in the system due to their architectural peculiarities.

Digital data is vulnerable

In the traditional accounting model, digital data is highly vulnerable. Whoever has corresponding access to the database (e.g., super admin) poses potential risks by default because he can hiddenly substitute the data and this fact will be exceptionally hard to verify and prove. For this reason, this party should be explicitly trusted by the system owner.

As a result, it is exceptionally hard to prove anything that happened in the system for both clients and system owners. The resolution of any kinds of disputes most likely involves the manual intervention of witnesses, courts, corresponding experts, and many more.

EXAMPLE

Imagine you wake up one morning, log into your banking application, and see that for some unknown reason, your balance is $900, while you are sure that yesterday it was $1000. However strange this may seem, such situations at times happen in banking systems.

Further, you open your card activity to make sure you didn't do any transactions since then, and it turns out that, indeed, no transaction was performed since yesterday. Meanwhile, you are confident that yesterday the balance showed you $1000.

The fact is that, however, most likely there actually was a transaction spending $100, but it was some days ago (not yesterday), and the transaction was pending for all these days and got confirmed later. This means that when you saw $1000, your actual balance was already $900;

it's just that it was not updated. An interesting point here is that it usually takes quite some time and effort for the bank managers to figure this out and make sure that everything is accounted for properly.

The reasons for such mismatches are that transactions in banking systems are predominantly non-atomic.

This is not the story about a bank trying to outwit you—it doesn't make any sense for the bank. However technically, the bank can do that, and if it is not using cryptographic signatures to sign each transaction, then the only possible ways for a user to prove that he saw different digits on the balance would be either by providing evidence to the court (maybe screenshot of the balance or witnesses) or by persuading the jury trial that it was really $1000. Both scenarios sound absurd. Or alternatively, the bank can sign each message via a digital signature, and this would make every operation provable.

Financial Internet presumes atomic transactions between separate accounting systems (for further details, see section 10.2), and that each operation (or balance inquiry) is genuinely signed via a cryptographic signature of its initiator and doesn't require people to verify its correctness.

First steps have already been made

The consequent result of the listed problems are some of the following risks and problems:

- *Difficulty of performing the system audit*
- *High fraud risk*
- *High prime cost of performing a transaction, resulting in high fees for an end user (especially for cross-border transactions)*
- *Uncertainty regarding the time a payment is accepted and received (sophistication of clearing and settlement processes)*

So, what does the complexity of performing the system audit manifest? Simply stated, an auditor cannot make sure that the data documented and provided to her is authentic and complete. This causes high risks both in terms

of intentional fraud by a payment system and, more trivial, unintentional mistakes in the accounting (that entail quite a few hassles even for the system owner to identify the reason of some mistake or mismatch).

The introduction of the two new concepts has already changed the situation for the better. These two concepts are cryptographic mechanisms for confirming transactions and the use of smart contracts to negotiate the conditions between the parties. Cryptography ensures the integrity and authenticity of all transactions (a digital signature cannot be half true or true only under certain conditions). The terms of any smart contract are known in advance and signed by the parties involved. No one can change or delete a contract after it has been confirmed; the contract is either fulfilled completely according to the strict predefined rules or is not fulfilled at all.

Some of the existing projects such as Ethereum have already successfully implemented both these approaches; however, these are permissionless systems, and as a result, they cannot meet other requirements such as the sufficient accounting system capacity, low transaction confirmation time, user privacy, independence of development while ensuring regulatory requirements (Figure 1.1). But the most important thing is that such projects presume the absence of responsible parties that do transaction processing, security updates, etc.—this is done by the "network" which behavior is not guaranteed.

If one considers only the business environment where censorship resistance and permissionless usage is not required, Ripple could be an example of an attempt to fulfil the above-mentioned requirements. However, this system has an internal currency that is needed to perform any operations (which is volatile, not backed by anything, controlled by one company). Also, one can't run the Ripple system on top of their banking infrastructure and interoperate with another bank that *hasn't joined* the Ripple network. Therefore, we receive another meta-system, which could be more efficient than SWIFT, but nevertheless limits the ability to interact with other networks because of an internal currency which is needed for each operation. If we look at how the Internet is built—it is a different approach—many networks can interoperate because they use the same protocols.

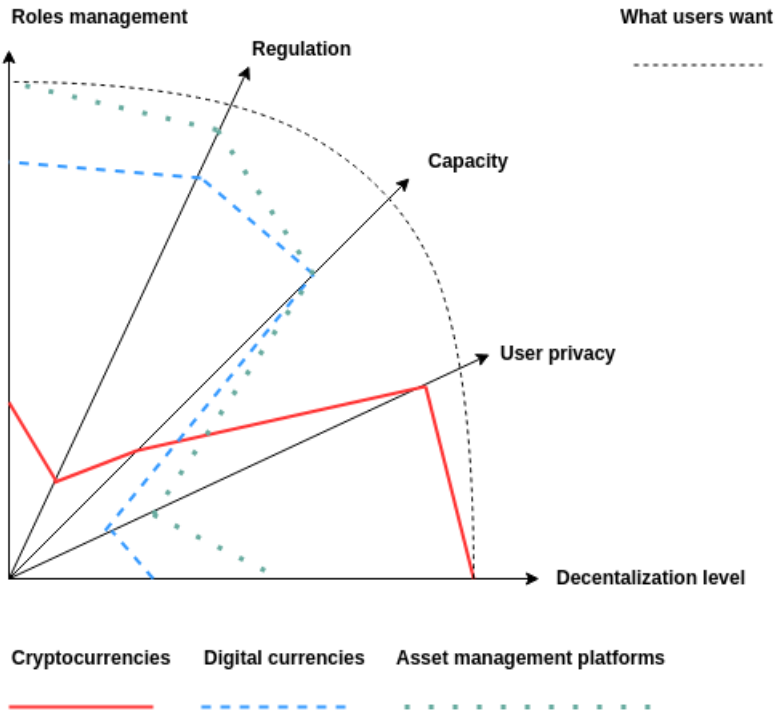

Figure 1.1 — Properties of accounting systems vs what users want

So, why does the presence of a base currency limit asset management? A base currency frequently impacts negatively the processes in the system since the price of this currency is unpredictable (while the developers of such systems pray that the price will grow). First, to conduct any kind of operations within the system, the user needs to pay fees in the base currency, meaning that regardless of the user's intention, she is always obliged to buy and hold a certain amount of this currency. In addition to that, not only the user is directly dependent on the base currency, but also all assets issued hold dependencies in such an accounting system, namely their price and a number of other parameters. Let alone the consensus in the system, which is often achieved in relation to the way the value of the base currency is distributed. Imagine what happens to ERC20 tokens if ether's price drops significantly and validators will no longer be interested in producing further blocks [3].

1.2 Main principles of Financial Internet

Financial Internet is a set of independent systems
that use the same language to communicate

The financial industry needs a unified and standard set of open-source protocols to share and store information about accounts (for easy and provable audit) and the settlement of respective transactions. These protocols should be able to support the processing of transactions that manage different types of assets, be compatible with identification systems and support the integration of external KYC providers as well as provide provable finality of trade.

In fact, Financial Internet should be a mesh network that consists of accounting systems, clients, data feeds and storages and links between them (Figure 1.2).

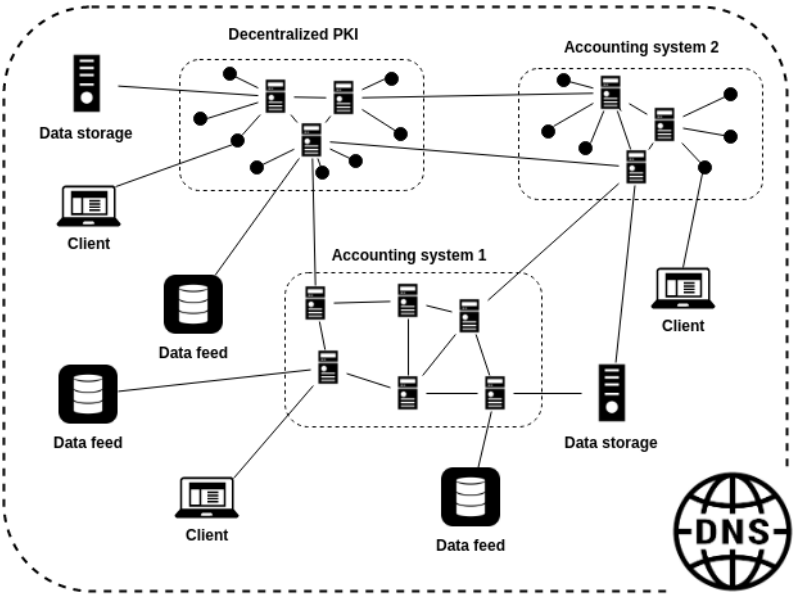

Figure 1.2 — Mesh network architecture of Financial Internet

The main principles of Financial Internet are as follows:

- Financial Internet is a set of **independent** accounting systems.

- Each system has a uniform API for the management of accounts, assets, and transactions.
- In the case of physical asset ownership rights digitization, there is always a custodian who is responsible for 1) safeguarding the asset itself and 2) providing the correct information for the accounting system. *In many cases, the custodian maintains the accounting system itself.*
- Every operation (e.g., requesting the account status, changing account details, initiating asset transfer, etc.) is cryptographically signed.
- Every data source is authenticated (has an identity) and responsible for what it signed.
- Systems may communicate with each other in real time to reconcile the data.
- Corresponding account relations are established ad-hoc and are based on the trust between entities such as financial institutions, clearing houses, merchants, users, etc. *There is no need to change existing trust relationships. It is just that they should be fully automated.*
- Financial Internet protocols and reference components should be a common good.

One interesting result - in the Financial Internet *an asset cannot disappear or, alternatively, appear out of nowhere in some accounting system* (as in the case of the Newton energy conservation law in physics and chemistry [4]).

New security standards

Accepting a new security model requires abandoning the traditional principles of interaction between the clients and accounting systems. Some of those to abandon are:

- ❖ *Password authentication mechanism*
- ❖ *Session engine (and cookies accordingly)*
- ❖ *Using services with a single point of failure*
- ❖ *Storage of sensitive data in an open form*

Passing a password through open channels and processing it explicitly by the authenticating party can result in an attacker gaining access to multiple systems (since most often, users have a single password set in different systems). Abandoning the password authentication mechanism does not mean refusing passwords. Instead, they will be used to generate encryption keys, and for the local users to access their own applications (primary authentication).

The sessions mechanism assumes the possibility of a number of attacks (more details covered in sections 8.2 and 9.1), which allow the interception of the session by an attacker and him gaining access while claiming to be the target user. To conclude, no more sessions means that users will be signing each request with their private keys.

Also, any accounting system should be fault tolerant. This property is quite difficult to ensure if such a system is supported by a single hub. Decentralization of accounting system processes is one of the key methods to ensure system resilience.

The last pressing issue is the storage and transmission of personal data in an open form which contradicts the GDPR. Encrypting data (predominantly by its owners) during its transfer and storage is a very important aspect without which building the Financial Internet would be impossible.

Transactions are signed by all parties involved

A transaction is considered final when all the needed signatures are present. Once all parties are identified in line with the needed requirements (such as KYC / AML) and they sign the transaction, the signatures are legally binding. All balances are adjusted in each ledger asynchronously.

Browser for assets

We can access any website using any browser installed on any device.
And it will look the same

An internet browser is an application (Figure 1.3) that allows a user to access information and services provided by application servers. An internet browser serves as a good analogy since regardless of which browser you use (Chrome,

Safari, or, God forbid, Internet Explorer, etc.), you can easily surf the internet and access any website.

Figure 1.3 — Traditional browser

Imagine a situation when you'd have to use different browsers depending on the website you want to visit. For any digital native, this is a true nightmare. In the same way, in the near future, people will remember the times of using different applications for different banks, payment systems, and stores (e.g. loyalty systems) as a bad memory (Figure 1.4).

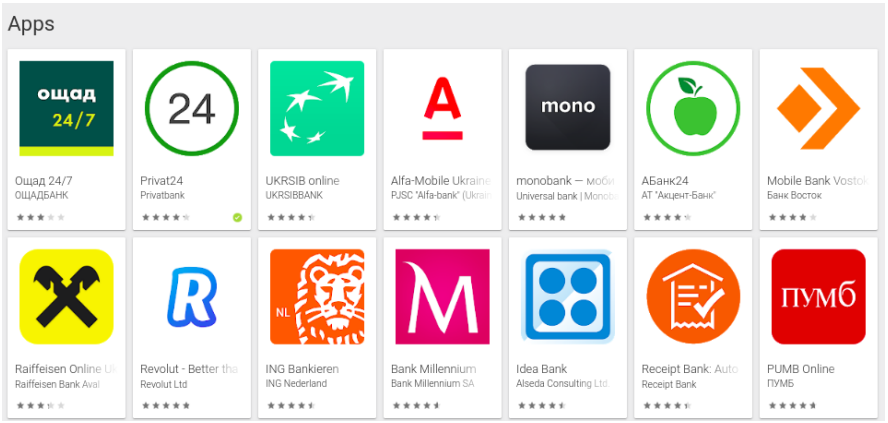

Figure 1.4 — It took years for thousands of applications to reimplement the same functionality

Financial Internet presumes that:

- Users can install one of the available applications on any device whereby access **any** system where a user has an account in. The main task of the application is to create and sign requests with the user's digital signature.
- There is no intermediary between the user and the accounting system since the APIs are open (as the PSD2 directive [7] says).

Financial Internet can work in a similar way, meaning that the user has her "browser" which she uses to access the services of various financial institutions, payment systems, etc.

It is noteworthy that the processing of keys and signing of transactions is performed directly on the client's device. All accounting systems work in a similar way: there is a client that creates and sends requests to the service, and there is a server that, in turn, confirms these requests.

The more compatible these systems are and the more uniform the APIs they use for communication are, the fewer hassle users will experience when conducting payments and transmitting votes using their browser. Eventually, this will lead to a situation where users feel no difference between using a browser to scroll through and to chat on Facebook and using a special browser to conduct financial transactions (Figure 1.5).

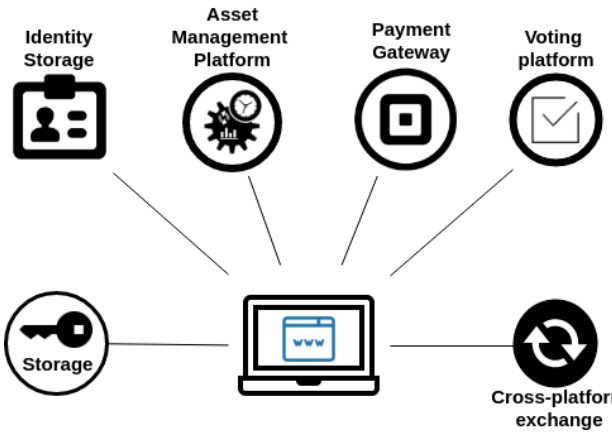

Figure 1.5 — Financial browser interacts with several accounting systems

1.3 What Financial Internet is not

Every business has its own accounting system. This happened for a reason

To better understand the idea and features of Financial Internet, we should first consider what it is not. Interestingly, the blockchain hype period has put quite a few stamps regarding the vision for the future. Some of them are as follows:

- *Global shared ledger*
- *Designated validators*
- *Unified, global consensus*
- *Base "native" currency*
- *Global identity provider*

No global shared ledger

Every accounting system (and the corresponding custodian) has its own ledger. Assuming that the creation of a global ledger as one single digitized source of truth that "documents" and manages the vast majority of processes and objects in our world is a viable solution (Figure 1.6), is a fairly widespread assumption, yet fundamentally wrong [5] (at least, this is our view).

One of the primary reasons it is not possible is that each system has its own requirements which have to be met and considered when building the system. Hence, the idea of a unified accounting system that would simultaneously satisfy all the requirements (e.g., high capacity, high level of resistance to censorship, absolute privacy of users, strict regulatory requirements in accordance with the current regulatory and legal field, etc.) is not possible to implement. It would simply render itself meaningless as a single system that simultaneously provides information on both financial transactions and user identifiers and voting results.

A more realistic approach (and the one that will be applied eventually) is the creation of a large number of independent systems, each with its own purpose, performing strictly a set of needed operations, and only containing information about a predefined set of entities (Figure 1.7).

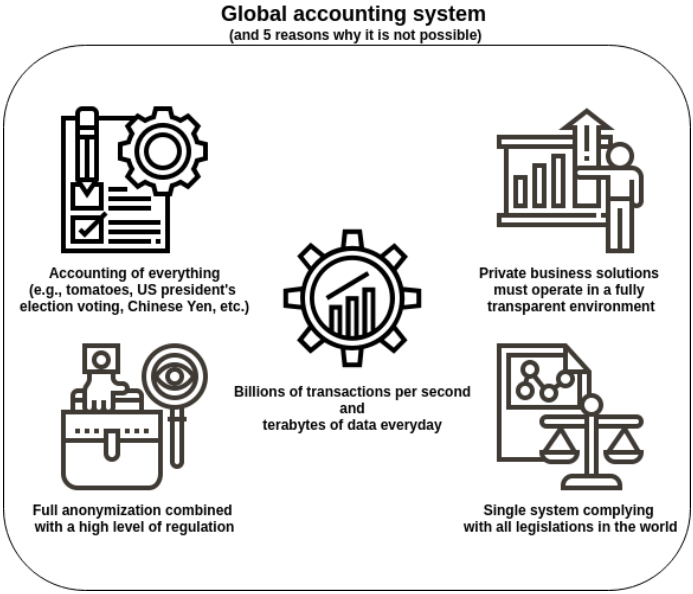

Figure 1.6 — Reasons why a global ledger for everything is a bad idea

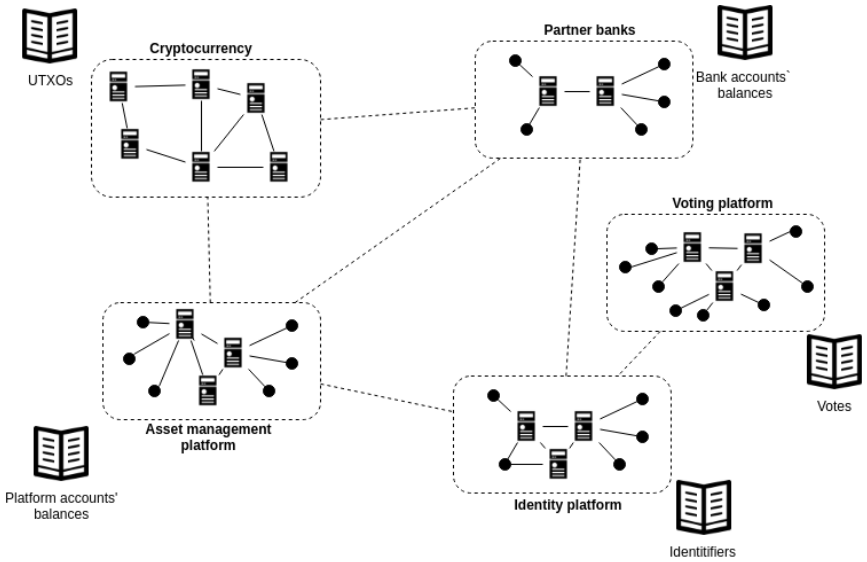

Figure 1.7 — Every system has its own ledger

What is left to be done here, is to organize bridges between these systems and their integration into one global network, Financial Internet. Note that ledgers can be made both private or public as well as shared only between certain entities.

No designated validators

Each platform independently defines the number of validators needed in order to function properly (and the way these validators reach consensus with each other). For example, in cryptocurrency systems (e.g., Bitcoin), every single person on the planet can be a validator (at that, unanimous); they do not limit users in terms of requiring any kind of privilege to verify transactions in the system.

This obviously is not the way it works in the business world where only involved parties validate transactions. Right now there is a limitation that requires parties to trust a particular system, but the use of consensus protocols between independent parties fixes it. The consensus mechanism needs to be permissioned by design; validators must be verifiably known (this is reached through digital signatures which are used when exchanging messages between participants). Also, such systems very often require the presence of a regulator who may not be involved in the process of reaching consensus, but it should nevertheless be able to store the entire transaction history and provably control the legitimacy of validators' activities to ensure that they do not violate any protocol rules.

No global consensus

As mentioned above, each accounting system requires a specific method of reaching consensus that meets its requirements (consensus is reached only between parties involved in the same accounting system). Depending on the requirements for different groups of entities (level of trust towards validators, requirements for the system capacity, etc.), different consensus mechanisms are used. Effectively, each group becomes a separate decentralized system with its own shared ledger and consensus among parties is reached in line with their requirements (Figure 1.8).

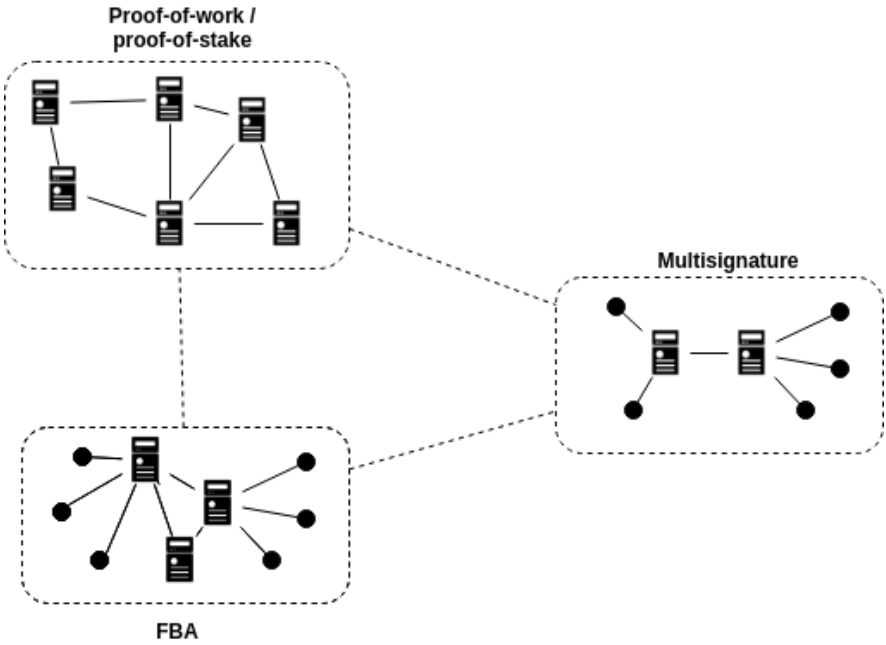

Figure 1.8 — Every system has its own consensus mechanism

No built-in coin

All digital assets are backed by custodians (it could be a financial institution, a warehouse or a licensed company). There could be a situation where the world selects one asset to be a global currency (it should be then serving all functions of money: storage of value, medium of exchange, and unit of account). Selecting the set of such currencies/assets could be a difficult task. But having a "protocol currency" independent from an accounting system seems to be a very bad idea, especially if its monetary policy is not aligned with the underlying economy. In general, assets can be stored in different ledgers (i.e., in different accounting systems), meaning that there is no some kind of a unified accounting unit (some abstract "unicoin") between these ledgers (Figure 1.9). The transfer of value between these platforms would be then implemented through the atomic swaps (which is very similar to the currently used *correspondent accounts* method).

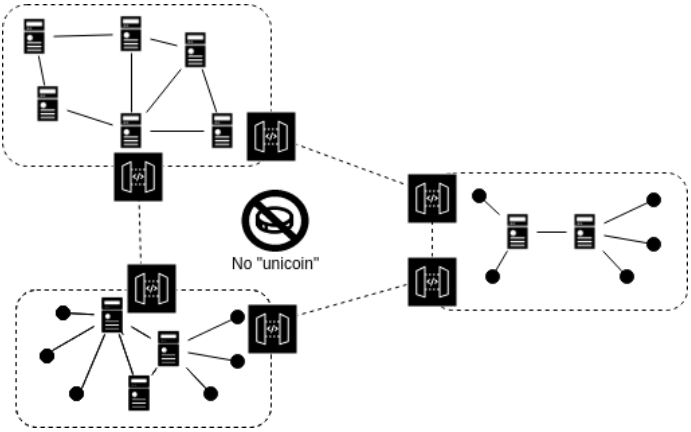

Figure 1.9 — Every system has its own accounting unit(s)

No global identity provider

Having a single identity provider would be a threat to privacy and freedom. Instead, Financial Internet should adopt some kind of *Web-of-trust* [6] approach where independent KYC providers synchronize data between each other for the purpose of preventing fraud (Figure 1.10). All requests and responses to identity databases are signed.

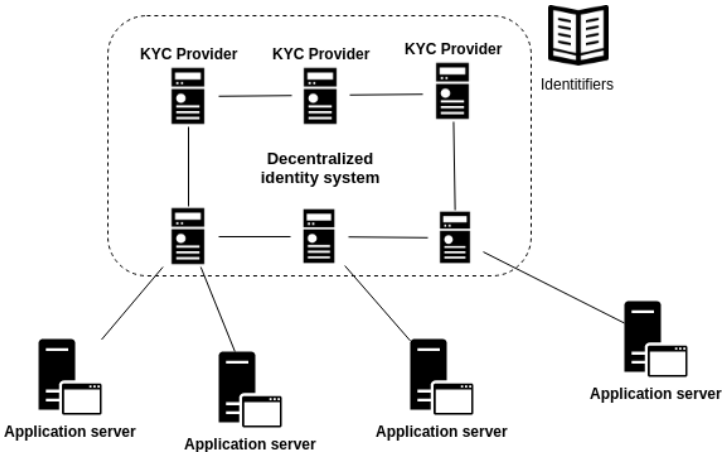

Figure 1.10 — Separate registration modules organize a common identification system

Global network of thousands of compatible accounting systems

Hence, Financial Internet is NOT a uniform distributed network. It does not consist of individual nodes participating in "one big thing". Large financial institutions can be both validators (or auditors) in several separate platforms, meaning they can independently carry out identification of end users (be a participant of a decentralized identification system) and support several validator nodes in digital currency systems (Figure 1.8). It is important to provide open and unified specifications that facilitate the development of compatible applications.

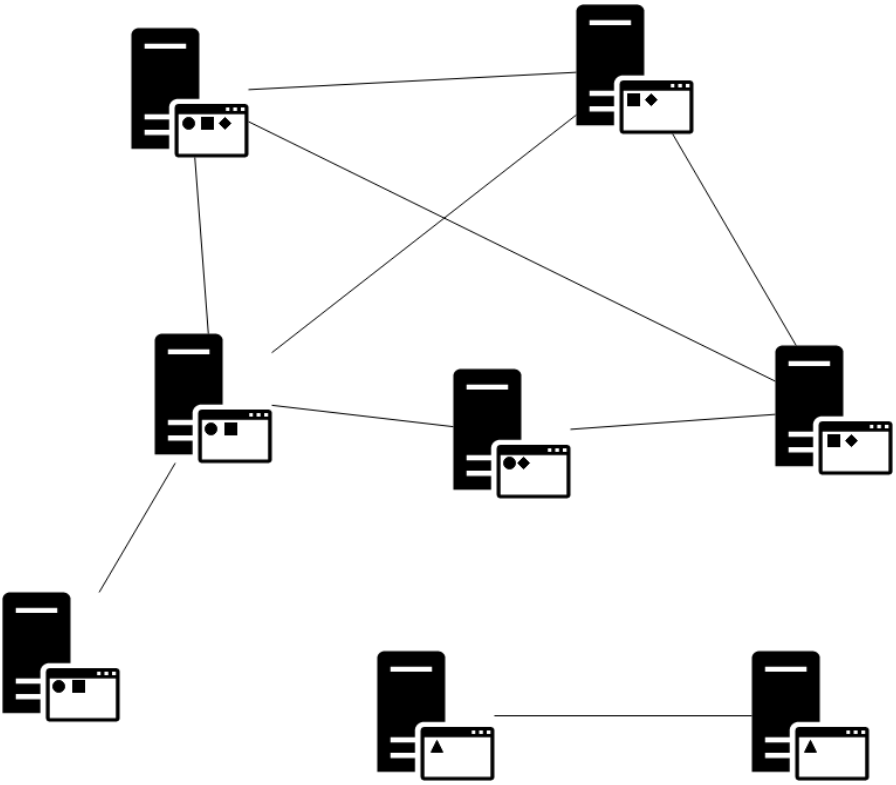

Figure 1.11 — Some nodes can be members of multiple systems

1.4 Architecture of Financial Internet

Components of Financial Internet are like Lego pieces

- *Accounting systems*
- *Client applications for accounting system*
- *Digital identity infrastructure*
- *Infrastructure for storing private keys*
- *DNS infrastructure (to provide the connection between the user and the accounting systems in which she has an account)*
- *Data feeds / Oracles*

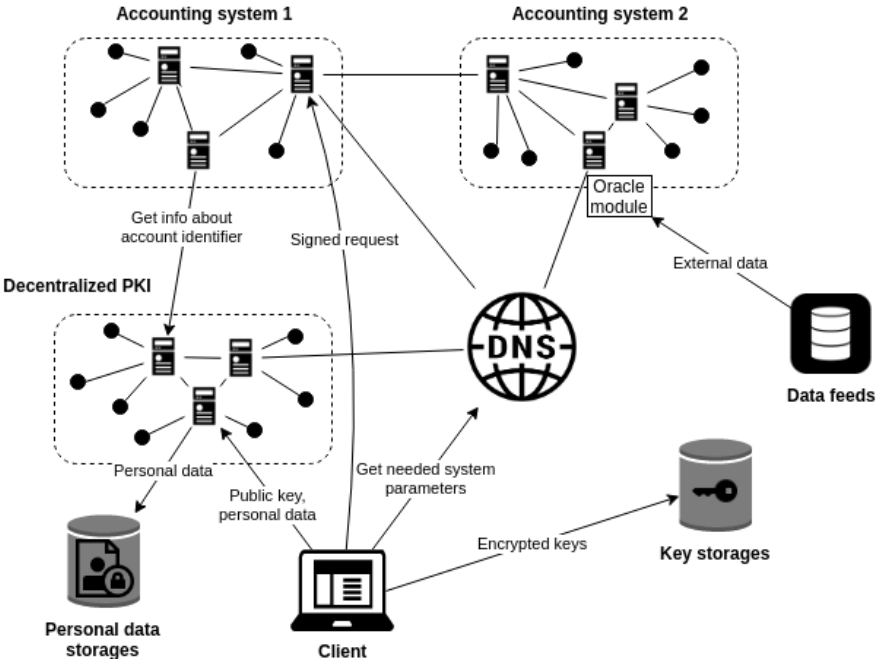

Figure 1.12 — Communication between Financial Internet components

Some of the most essential features Financial Internet provides include the following features:

- Integrity of account balances
- Provenance of assets
- Interoperability between components
- Secure synchronization between accounting systems

Accounting systems

Any accounting system has its own purpose per se. Different types of accounting systems exist, some of which are:

- ❖ *Asset management platforms*
- ❖ *Voting platforms*
- ❖ *Crowdfunding platforms*
- ❖ *Payment systems*
- ❖ *Warehouse receipts systems*
- ❖ *Public registries*
- ❖ *Shareholder registries*
- ❖ *Cryptocurrency and digital currency systems*

The general purpose of accounting systems is to provide tools for managing asset ownership rights that belong to users. These systems support processes such as issuance, transfer, exchange, and withdrawal of assets. Further, they are the primary sources of information about any asset balance within one system. In conclusion, accounting systems enable the management of digitized assets. For more details about asset management, refer to section 4.

Types of asset management systems

Let's define what kind of tokenized assets exist and analyze the features of each.

- ❖ *Cryptocurrency*
- ❖ *Central bank currency*

- *Digital currency*
- *Commodity-backed tokenized assets*
- *Equity tokenized assets*
- *Digital collectibles*

A *cryptocurrency* is an independent digital currency where the management of the following processes is decentralized: currency issuance, confirmation of transactions, data storage, accounting system audit, and decision-making regarding the system updates. Note that in such accounting systems, the number of validators is unknown, the validators are anonymous and have no reputation. The mechanism for achieving consensus in such systems should be as decentralized as possible, and it is currently mostly based on PoW, PoS [8].

A *central bank currency* is a system with all processes (governance, custody, issuance and distribution, transaction processing, audit) centralized which is managed by a central bank, whereas digital currency is pegged to a fiat currency in a 1:1 ratio.

In *digital currency systems,* processes such as validation of transactions, fee configuration and updates can be handled by the decentralized network of users, while the currency issuance and initial distribution are managed by a centralized organization.

Commodity-backed tokenized assets represent ownership rights in a particular amount of commodity. A tokenized asset is managed in a system with centralized governance, custody, and issuance. These processes are performed by the service provider or by the custodian of a physical commodity. One tokenized asset is always backed by a fixed amount of commodity, and the 1:1 ratio is guaranteed by a designated party. Examples are the U.S. dollar prior to 1971, which was a token that represented gold, or warehouse receipts—in these cases, processing and audit are done in a centralized way as well [9].

A *security* is a fungible financial instrument that represents some type of financial value. Tokenized security assets represent the ownership of an underlying security or a share in a cash flow generated by the system. Assets are managed in a system via centralized governance, custody, and issuance. These processes are performed by a depository or by a company itself. One token

always represents a certain number of shares or a percentage of the cash flow. Processing can be done in a centralized way by a depository.

A *collectible* is any object regarded as being of value or interest to a collector. These objects are unique and non-fungible. One token represents the ownership of a particular object. Governance and issuance are centralized.

Besides the mechanisms to merely account for one's assets, voting platforms are needed to take decisions (regarding literally anything: from the U.S. presidential elections to choosing the color of the corporate T-shirt). The purpose of the voting platform is the reliable accounting of user votes whilst ensuring the integrity of transactions and the transparency of voting results. Now putting these elements together, such an integrated system enables reliable and provable digitized decision-making for users. The detailed principles of operating a decentralized voting system will be discussed in section 9.3.

Digital identification infrastructure

Besides accounting and voting mechanisms, user identities need to be verified prior to the asset transfer within any platform. Digital identification systems are the source of information about user identifiers. In these systems, users are registered in order to establish a relationship between the final subject (e.g., actual U.S. citizen or, in the case of IoT, a thermometer on the board of the freezer) and its identifier. As a result, these systems provide a digital identity. We will discuss the principles of building decentralized identification systems in more detail in section 7.

Client applications

Client applications send signed requests to accounting systems and process the information received from them. It is important that applications are implemented by multiple independent manufacturers and are open-source.

In addition to regular user wallets, there may be special applications aimed at automating a number of specific functions. The typical examples are trading applications, applications with the built-in atomic swap functionality, applications with a payment channels mechanism implementation, etc.

Key servers and data storages

All of the systems represent the components of Financial Internet. However, in order to perform certain actions on the asset management platform, participants are required to provide additional data. Users need keys to perform actions on the platform and the keys need to be stored safely.

A key server stores the user keys in an encrypted form and provides access to authorized persons only. The key server functioning principles will be described in more detail in section 8. Beyond the key storage, data warehouses are needed to store additional data that is mandatory in order for transactions to be executed (e.g., documents, media, etc.)

Data sources / Oracles

Determinacy is one of the crucial features of an accounting system. This means that transactions in a particular system can rely only on the data that exists in it. This entails coherent and distinct decisions but, at the same time, limits the flexibility of configuring contracts. To ensure the availability of transactions based on the data outside the boundaries of an accounting system, the corresponding data sources must be provided. These data sources are called oracles.

DNS infrastructure

Financial Internet, not unlike the usual Internet, cannot go without the DNS infrastructure, whose basic principles essentially won't be changed but rather extended to operate in a decentralized manner and provide a way for accounting systems/merchants to compete for the data that will be shown to the users. Thus, there should be an infrastructure that:

- Tells the client application which systems it is connected to;
- Helps to find IP addresses of its validators and auditors;
- Sends push notifications from the accounting system to an application;
- Notifies both parties if there was an identity change;

Roles and applications

We will single out the following roles and also determine which of the components described above fulfill these roles (to have a clearer understanding, refer to Figure 1.12).

1. Data provider (DNS, data sources and oracles, digital identity infrastructure);
2. Data storage (key servers and data storages);
3. Data processor (accounting systems, identity infrastructure);
4. Data consumer (clients, accounting systems);
5. Data protector (data storages, and in the context integrity, accounting systems and DPKI).

Regarding the applications, the most important principle is that all actions are signed with a private key and that the corresponding public key/identity has required permissions. The software doesn't really matter a lot in such a case.

Processes

Now let's consider some examples of the processes that connect the most important objects of financial relationships, such as accounting systems, identity providers, user applications (noteworthy, earlier many of these objects couldn't operate in a single shared infrastructure). So, examples of such processes are reconciliation, data transfer, and search.

Reconciliation is an accounting process used to determine whether the money leaving an account matches the amount spent. During this process, assets do not leave the boundaries of their accounting system, and only the owners of these assets are changed. In section 10, we describe the main approaches of how the ownership transfer between separate accounting systems is performed using cryptographic methods (we also cover the architecture of modules which allow integrating with external payment systems that can't use the reconciliation approach).

Data transfer. Since accounting systems can only rely on the data they account for, it is necessary to ensure the reliable transfer of data from within the accounting system. This is done via oracles who are trusted intermediaries

furnishing external data to the accounting system. Each data feed is a separate transaction signed by the oracle providing it. To increase the level of objectiveness of the received data, it is recommended to increase the number of independent oracles furnishing it.

Search. As in the usual Internet, users of Financial Internet should be able to search for accounting systems using clear and human-readable names. And this is the purpose of DNS that will provide a kind of an address book for the accounting systems. Hence, a user will enter the name of the required system into the application (browser) and get the corresponding network addresses.

2 ACCOUNTING SYSTEM ARCHITECTURE

Regardless of which information an accounting system hosts—gold, money, tomatoes, or sand—it has the uniform core, modules, and APIs

An accounting system separately stores and manages accounts, assets, transactions, data, and rules of their management layer

Each database has its own APIs (built uniform) and can use a decentralized architecture

2.1 Accounting system components

As shown in Figure 2.1, an accounting system can be divided into two levels.

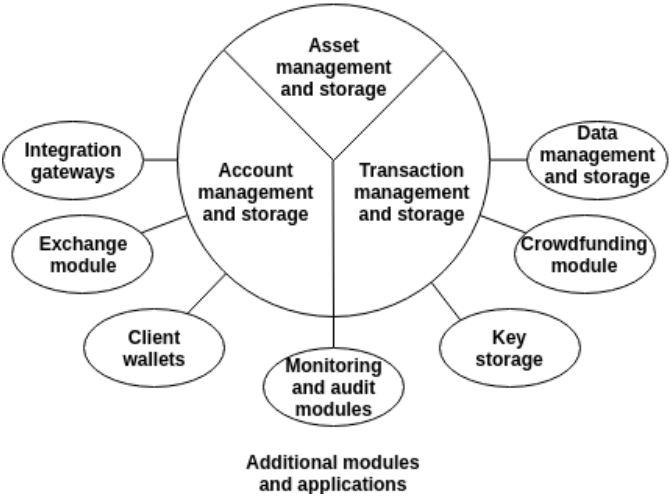

Figure 2.1 — Two layers of the accounting system

The first level consists of ledgers, which store information about accounts, assets, and transactions as well as the rules that govern their behaviour. All the data can be shared with all validator nodes. Thus, eliminating one validator node from a system cannot result in the loss (or inconsistent change) of the ledger state.

The second level consists of additional modules that connect ledgers with

external systems. These modules are as follows (Figure 2.2):

- ❖ *User wallets*
- ❖ *Administrator applications*
- ❖ *Accounting system audit applications*
- ❖ *Monitoring and reporting systems*
- ❖ *File storages*
- ❖ *Key storage servers*
- ❖ *Payment system integration modules (banks, payment gateways, other external accounting systems)*
- ❖ *External system integration modules (identity services, notification services, etc.)*

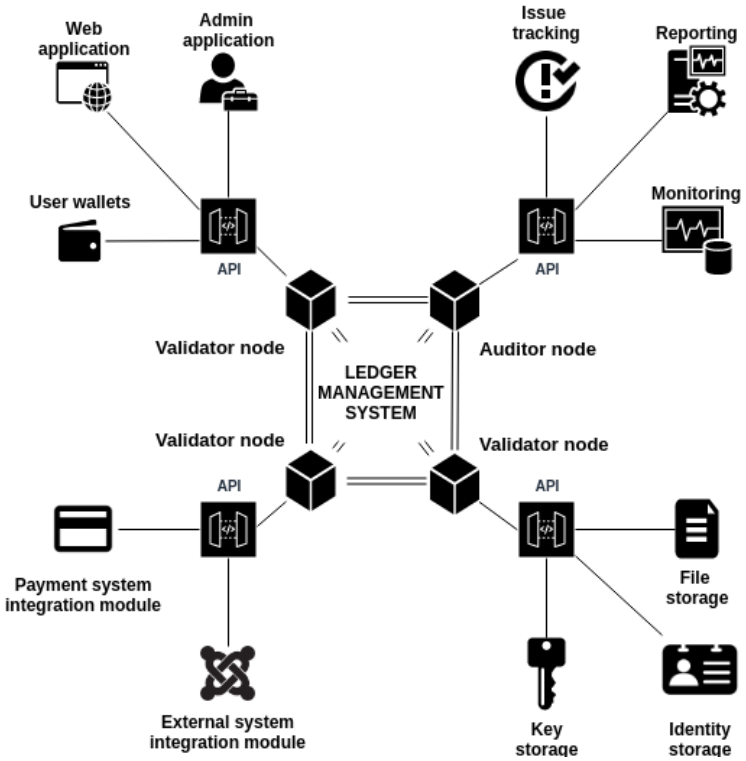

Figure 2.2 — Additional modules of the accounting system

Ledger

As noted earlier, the main task of a ledger is to provide relevant information about accounts, assets or transactions (Figure 2.3). Each database has its own interface and can have a decentralized architecture, meaning that its state can be stored on different servers and synchronized in real time.

We explicitly state that accounts, assets, and transactions can be stored in different ledgers as well as rules that govern their changes. This is necessary to independently improve and update the codebase and, if needed, use components from different suppliers.

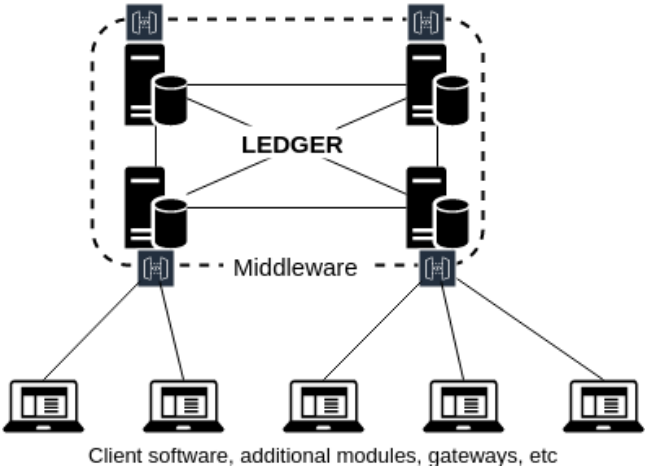

Figure 2.3 — Ledger is the main component of the accounting system

Middleware

Middleware is an intermediate software layer (between validator nodes and additional modules) that performs preliminary verification of transactions and sends them to validator nodes and retrieves the current system state and history of accepted transactions.

Before network nodes receive transactions, they are processed by middleware. Middleware is a "pre-validator" for system errors (which are not related to protocol errors). An error message will contain information describing why the request cannot be completed successfully. If the transaction

was not correct, middleware returns the appropriate error message and rejects the transaction. Here are some examples of errors:

- Middleware cannot understand the request due to invalid parameters
- Middleware is not synchronized with the current state of the system
- User is not authorized to see corresponding data
- Middleware cannot give an appropriate result for the user's request
- Request method is not supported by middleware
- User sends too many requests to middleware within one timeframe

In order to determine the reasons for rejecting transactions by middleware, the following transaction attributes should be analyzed: type, status, and error details.

User wallets

Wallets are client-oriented applications that provide a wide range of functionalities from storing encrypted private keys on a device to transferring, withdrawing, and trading digital assets. They interact with the system core directly through middleware (see further details below) and sign both transactions and requests locally. Such an approach ensures the security of users' private keys, even in the case of MITM (man-in-the-middle) attacks.

Admin applications

A platform administrator is a role that can perform the most diverse range of operations: from managing user permissions and confirming particular operations (e.g., issuance, deposit, or withdrawal) to even managing permissions and operations of other administrators. For this, administrators use specialized software that supports the set of operations required for each administrator.

Auditor applications

An auditor ensures that transactions on the platform are performed according to the protocol rules. The main function of an auditor is to store the full history of ledger(s') changes and verify the integrity of any system operation (that validators haven't agreed to roll back or change the ledger state) as well as

to validate accepted transactions and thus prevent double-spending and mismatch of user permissions (with respect to the underlying operations).

Monitoring and reporting applications

In addition to auditors, an accounting system also contains monitoring and reporting applications that are associated with specific events in the system. The purpose of such applications is to generate statistics for a particular business based on individual operations and associated accounts.

Files and key storages

Storages are additional modules designed for the convenience of working with the platform. Data storages are used to avoid placing all the data/metadata in the ledger but rather to only place the corresponding hash value. Key servers are used for convenient and secure access to keys—from any device and at any time (without the need to transfer them manually).

Payment services integration modules

A payment services integration module is a set of modules that play the role of a gateway between an accounting system and cryptocurrencies' public systems, banks, payment gateways, exchanges, etc. It enables operations such as deposit, withdrawal, changing the exchange rate in another system, and more.

External systems integration modules

An external system integration module is a set of components that interconnects an accounting system with various external systems. It is responsible for a wide range of functionalities from sending notifications to automating the user identity verification.

Main entities handled by the accounting system

The accounting system deals with several base entities (Figure 2.4):

- *Account*
- *Asset*

- *Transaction*
- *Data*

Account is a protected data structure that contains information about a set of balances for certain assets which are managed via cryptographic keys. Accounts also contain permissions of corresponding users within the given accounting system.

Asset is a data structure that represents parameters of some physical or virtual object.

Transaction is an authorized data structure that carries instructions in terms of changing the state of accounts or of the entire system.

Data is a piece of information which is unstructured in terms of accounting (e.g., pictures, memos, etc). Data is attached to an account (their hash value) and can be used to expand its capabilities.

Figure 2.4 — Entities and components that work with them

In sections 3 to 6, we will discuss how these entities are stored and handled as well as their lifecycle and useful functionality.

2.2 Accounting system roles

Roles in the accounting system are defined by its owner and represent existing business processes

Depending on the roles performed on a platform, participants use different software. Among the most common roles within an accounting system are:

- *Administrator*
- *User*
- *Validator*
- *Auditor*
- *Regulator*
- *Data custodian*
- *Physical assets custodian*
- *Oracle*

In this section, we will describe who assigns and performs the listed roles.

NOTE

There are also specialized applications used by system members for specific operations. For example, applications with a special interface for merchants, traders, etc. Obviously, the owner of the accounting system decides himself which roles to create.

Administrator

Administrators deal with the management of an accounting system in general. Administrators are assigned by the platform owner and are responsible for confirming user requests, configuring permissions, setting fees, and monitoring how integration modules operate. At the same time, it is also possible (and preferable) to differentiate administrators' permissions for managing different functions (e.g., one admin manages withdrawal requests, another sets fees, etc.) and even to mutually manage one function that is critical

(e.g., the operation for asset issuance is distributed between 3 administrators). Such a risk diversification is possible to achieve through the multisignature mechanism.

User

Users can use their own software to manage their own account (control their assets, manage identifiers, set limits, etc.).

Validator

Validators perform the key role in the accounting system: they approve the change of the ledger state. A validator stores the entire transaction history and reaches consensus on its updating. If the system is centralized, the validator confirms the transaction alone. If an accounting system has several validators, then they have to reach consensus regarding the transaction set that will be the base of an updated state.

Auditor

Auditors monitor and control operations performed in the system. Auditors maintain a full copy of the system, which allows them to monitor that transactions were processed according to actual protocol rules and enables them to be an additional source of truth in addition to the system validators. An auditor synchronizes transaction history with validators and other auditors.

Regulator

Regulators are very similar to auditors, but they have the power to enforce the rules that they set.

Data custodian

Data custodian is the role that is responsible for storing data that is not recorded on the ledger.

Physical assets custodian

Physical assets custodians store physical assets that are not (and primarily

cannot) be managed in a digital form. The goal of the physical assets custodian is to store whatever represents a respective asset accounted on a platform. Custodians implement the important function of providing reliable information about the amount of asset stored. They have a digital identity and produce an authenticated data stream. A custodian also needs to run a full node and compare the current amount of assets accounted on a platform with their actual physical backing.

Oracle

An oracle is a source of information about events that occur outside the accounting system [13]. If transactions in the accounting system require any external data, then an oracle provides it. The more independent oracles that provide the data to the system there are, the more objective the data recorded in the system can be considered.

2.3 Processes in the accounting system

Processes in accounting systems can be divided into self-governance as well as into storage and management of accounts, assets, transactions, and data

These processes will be described in detail in the following chapters. In this one, we will consider the main entities that relate to them.

Accounting system processes are as follows:

- *System management (governance)*
- *Asset storage and management*
- *Account storage and management*
- *Data storage and management*
- *Transactions storage and management*
- *Accounting system audit*

In this section, we will consider the basic features of managing the enlisted entities and analyze which system components are involved in the before mentioned processes.

Accounting system management

The basic processes for system management are configuring the communication between its components and updating the accounting system protocol.

Configuring the connection between system nodes

If the system has only one validator, all that is needed is to provide the connection of this single validator node with additional modules. For secure interaction, these modules are assigned key pairs to sign the messages as well as to encrypt them for the purpose of confidentiality. Hence, the validator confirms transactions and updates the ledgers state, and all that is needed for reliable system operation is to properly handle requests received from clients and modules.

If there is more than one validator, then these validators need to mutually reach consensus regarding the ledger state. The solution is now more complex: validators need to properly organize the connections between each other in order to correctly process messages. Every message exchanged between validators must be signed by them to preserve the integrity and authenticity of messages.

Protocol versioning

A protocol version is an integer number stored in the header of every transaction set (the protocol version that was supported when the next set of transactions has been added). A version number is incremented with every protocol change. Transactions and operations that initiate a change in the ledger state are processed according to the specified protocol version (Figure 2.5).

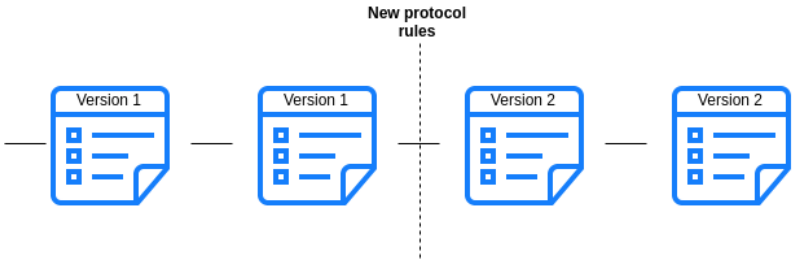

Figure 2.5 — Protocol versioning

Notably, the protocol itself does not keep track of which version the nodes should support; instead, this is what every node owner does independently. Each node has its own values of the supported protocol versions as well as the version blacklist (the list of versions that are not supported by a specific node) [10].

Ledger objects versioning principles

An accounting system contains several types of objects. The basic ones are accounts, transactions, and ledger states.

Accounts are the main data structure of an accounting system. Every account has its own identifier that specifies its permissions (for more details, see 3.1).

Transactions are commands to change the ledger state. Every subsequent ledger state corresponds to the subsequent set of transactions (we will consider the transaction structure and how transactions are performed in section 5).

A ledger state is basically a table that stores the accounting system state. This state is the result of a set of accepted and performed transactions from the moment the system was launched until a specific time.

There are two ways of how objects can be updated, through a softfork or a hardfork [11]. A softfork is possible to be implemented if the structure of the updated objects contains extension fields. They allow introducing extra fields without having to create a new type of object in the system. If an object supports such fields, then it can be updated with a softfork. The softfork update implies that the new data structure is supported by both the updated nodes and the nodes that have not yet been updated. This also means that updated nodes will not accept an old-format data structure (objects are presented in Figure 2.6).

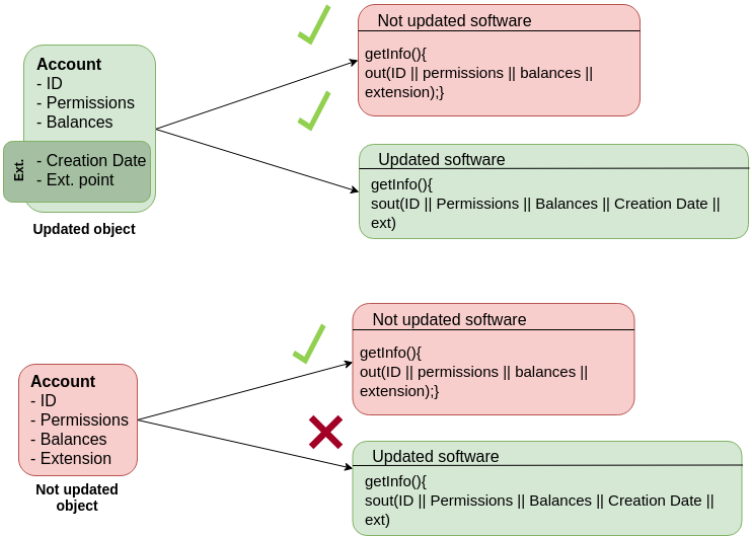

Figure 2.6 — Process of object updating

If an object does not contain extension fields, it must be cloned (i.e., an object of a new type must be created) and further extended. The update must then be performed through a hardfork. Hence, the non-updated software will not be able to process updated ledger objects.

Operations versioning principles

Operations are versioned by adding a new operation to an accounting system. That is, if you need to add a new parameter or modify an existing one, the version will be updated by creating a new operation (Figure 2.7). This allows avoiding errors when interacting with non-updated objects but adds some redundancy.

```
accountCreationProcedure (int version) {

case 0:
        createAccount(ID, timestamp, roleID);
case 1:
        createAccount(ID, timestamp, roleID, meta)
....
}
```

Figure 2.7 — Operation versioning

Overlay versioning

On the level of message transfers between nodes, the cloning model of broadcasted message types is used for versioning similarly. That is, every modification of a message must be followed by the creation of a new message type. Every node independently determines which message versions it can accept. The initial message that peers exchange when connected to each other, contains the minimum and maximum versions supported. One of the parties can instantly disconnect if it is incompatible (Figure 2.8).

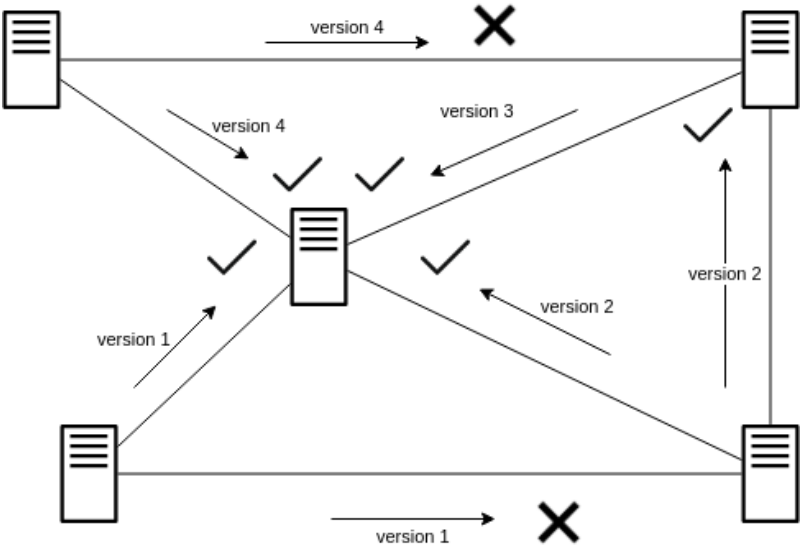

Figure 2.8 — Overlay versioning

Asset management

Asset lifecycle management is another important process in the accounting system. This process addresses not only asset owners but also custodians, administrators, and end users. Asset lifecycle and the features of asset management will be described in more detail in section 4.

Account management

Another process which is of great importance in the accounting system is the user management, more specifically the management of account permissions. A crucial role in this process is played by platform administrators. Every update associated with the user account must be initiated by a corresponding transaction signed by an administrator who has the appropriate permissions.

To manage assets, every account in the accounting system has a set of permissions. These permissions are checked during the stage of transaction signature verification—that the corresponding public key is bound to a particular identifier and its permissions.

Transaction management

Transaction management processes include signing and verifying transactions as well as updating the ledger state based on the transaction data. While signing the transaction is related to the initiator (or initiators) solely, the transaction verification and its confirmation involve system validators. These processes will be discussed in more detail in section 5.

Accounting system audit

To perform an audit of the accounting system, an auditor must connect to the validator(s), download the entire transaction history, and verify it according to the protocol rules to get its final state. Having the final state, the auditor sends a request to validators to receive their state and compare it with his (basically the state he receives is the same as all clients receive, but in this case, he has full transaction history downloaded and can make sure that the state is actually valid).

Compared to traditional approaches, such kind of audit is more automated and reliable. Moreover, most often it is performed in real time. If the accounting system operates in a decentralized environment, then the auditor must join the network, i. e., deploy and maintain an auditor node.

The key difference from traditional approaches is that the rules describing how the protocol should operate are written programmatically and the system

works autonomously. Hence, all that an auditor needs to do is to download the transaction history and start to continuously verify the accounting system state using automated software. Compared to traditional accounting systems, automation is achieved through a different approach to data validation and storage.

In this case, the ledger by itself cannot be considered the primary source of information because it is the consequent result of transactions performed. This means it is specifically the performed transactions that are the primary source of information, and only having them, one can get the actual final state and assure oneself as to its objectiveness.

2.4 Accounting system functions

The main function of the accounting system is to keep ledgers of operations with all its entities (accounts, transactions, assets and their rules) immutable and accessible

Regardless of the accounting system's purpose, there is a set of functions it possesses. In this section, we will very briefly cover each of them and then introduce more details later on in the book.

- ❖ *Transaction processing*
- ❖ *Transaction history storage*
- ❖ *Data storage*
- ❖ *Governance (decision-making)*
- ❖ *Protection of data and processes from internal/external attacks and failures*
- ❖ *Asset accounting*
- ❖ *User permissions verification and management*
- ❖ *External systems incoming data processing*

Transaction processing

Transactions are processed in two stages. The first stage is performed by middleware. At this stage, several key aspects are verified: the correctness of

transaction structure, size, fee amount, presence of funds on the balance, etc. Hence, all system errors related to failures and denial of service must be eliminated at this stage.

The second stage of the transaction processing is performed by the validators. This stage presumes the verification of a transaction in terms of the protocol rules—that it is not trying to double spend assets, that accounts of transaction parties have the required permissions, that it is not conflicting with other transactions, etc.

After the transactions have passed the first and second processing stages, a validator (or a group of validators) decide to add them to the database and update the ledger state.

Transaction history storing

The transaction history is stored by the platform validators, auditors, regulators, and (if needed) by the data and asset custodians.

Validators must store the entire transaction history to reliably take decisions about updating the ledger state. Auditors and regulators store the history to verify the compliance with protocol rules in real time. Having access to the entire transaction history, asset and data custodians verify whether the actual volume of assets matches the data on the ledger.

Data storing

Personal data of a subject is stored by the data storage providers (and they are solely responsible for this data integrity and availability). Access to the data is in the hands of a user because if needed, data can be encrypted on the user's device and be sent encrypted to the data storage. Public data required for the system operation (which does not need to be confidential but rather available) are stored in a separate system module and managed by an appropriate administrator (or several administrators).

Governance

The accounting system is managed by its owner, regulators, and validators. The owner and regulators can adjust the protocol rules and offer updates.

However, an update cannot take place until the validators start to operate according to the new protocol rules.

Protecting data and processes from internal and external attacks and failures

Each role within the accounting system is responsible for the security of those processes it participates in. For example, the task of users is to securely store and manage their keys. Administrators deal with issues related to the correct operation of modules, distribution of user permissions as well as confidentiality and integrity of the stored data.

Asset accounting

Validators and custodians are responsible for the asset accounting processes on the platform. Validators verify the correctness of the transactions performed on the platform (e.g., that they do not try to double-spend assets, that accounts receiving assets have permissions to do so, etc.). The task of custodians is to ensure that the data on the ledger corresponds to the actual state of things (i.e., the amount of assets accounted on the ledger corresponds to the actual amount of the backing).

Managing and verifying user rights

User rights are managed by the platform administrators and verified by the validators during the transaction verification.

Processing of data from external systems

The responsibility for processing data from external systems (or data provided by oracles) lies with separate modules and the administrators who manage the modules.

2.5 Accounting system security

The main principle of system security is that every component and process must rely ONLY on the security of the private and public keys of the entities involved

One of the necessary conditions for building a reliable accounting system is the development of a security policy, which describes in detail potential vulnerabilities and threats to the system and formulates mitigation strategies to deal with adversaries, further referred to as so-called threat and adversary models [13].

A *threat model* is a structured description of possible threats: sources of threats, the information security services that are violated by each respective threat as well as potential methods of threat implementation.

An *adversary model* is a structured description of a potential adversary: category, goals, technical skills, and qualifications.

First, let's consider who can be or become an adversary in the accounting system. An adversary may be an individual or a group of individuals who perform actions that violate the system's security policies. Any of the below-listed can fall into the category of adversaries:

- *Platform users*
- *Validators*
- *Administrators*
- *Auditor*
- *Regulator*
- *Identity provider*
- *Software/hardware developers*
- *Internet services provider*
- *Staff serving workstations (servers)*
- *Manufacturers of wallets and/or complementary software*
- *Data and assets custodians*
- *Oracles*

What might the adversary's potential goals be then? First, it may be modifying the database and the state of the users' balances. Then, it may be restricting access to the current database state for a certain node or a number of nodes to obtain permissions of administrators, issuers, etc.

To complete an adversary model, one needs to thoroughly examine the technical skills and qualifications of an adversary. The development of a CISS (complex information security system) implies that one must assume that an adversary has any kind of equipment (from a laptop to cluster), has access to the newest software and hardware as well as the required technical skills and qualifications to orchestrate an attack. The only thing that an attacker doesn't have access to is the private key of a user (under the assumption that it is stored securely). This is the basis for the security of all new generation accounting systems.

The following list shows the most common information security services that any given accounting system typically provides and that can be violated:

- *Confidentiality*
- *Integrity*
- *Availability*

Confidentiality. Depending on the purpose of the accounting system, it may not provide (or may partially provide) such security service. Some examples of confidential data in the accounting system are:

- Data about transfers, exchanges, and details of all transactions in the system
- Users' personal data
- Connection between users' personal data and data about their transactions and account states

If an accounting system is permissionless, then the responsibility for ensuring the confidentiality of data lies solely with the users. Most often, such systems do not require user identification, so the task of ensuring the confidentiality of personal data implies rendering it impossible for anyone to associate a particular public key (an identifier) with the data about its owner (a leak may either occur due to the user's carelessness or, for example, if the

software had a built-in vulnerability). At the same time, the task of ensuring confidentiality of transactions is implemented on the protocol level (e.g., Monero uses ring signatures by default, while Bitcoin has similar techniques which are optional).

Permissioned accounting systems often require user pre-identification, meaning that such systems feature additional steps to ensure security (according to GDPR [14]) when storing and processing users' personal data. Since the purpose of such platforms is to provide asset management services that are frequently subject to regulation, the issue of the confidentiality of user transactions becomes critical—transactions have to be open to appropriate administrators and platform validators for them to verify the necessary permissions with the transaction data still not available to adversaries; this, in turn, introduces new threats.

Integrity. The usage of blockchain technology in accounting systems is aimed precisely at ensuring integrity. Users must be sure that any performed transaction cannot be modified after it has been confirmed. These are the full nodes within a system (the nodes that store the entire transaction history) which guarantee transactions to be final upon confirmation. The issue of maintaining the integrity of users' personal data (if the system requires identification) has to be safeguarded by the platform owner. If the user is unable to store the full node, she can use the SPV technique and store only the data necessary for verification (if the platform supports it) [15].

Availability. If an accounting system is open to be audited by anyone, then everyone can access all data (transactions) and check the performance of all actions for updating its state. If a system is permissioned, then users must have access only to the allowed data and can only perform operations which they have permission to perform. Therefore, two problems must be solved by a platform: the verification of user permissions in terms of data access and access restrictions ensuring that a user is only allowed to access data she has been granted permission for. The level of accessibility can be increased by increasing the number of independent validators and ensuring that each of them supports the necessary middleware components for processing requests from platform participants.

Threats in the accounting systems

An accounting system has a strict set of roles which its participants can play (as opposed to permissionless systems, where any participant can be a validator, auditor, user, etc. all at the same time). Accordingly, violators may already have a certain level of influence on the accounting system depending on their role. Further, we will consider the possible threats from end users, validators, administrators, platform owners, etc.

Asset accounting threats

One of the most critical attacks in the accounting is the creation of a transaction (or a set of transactions) where certain assets are being spent twice or more (i.e., creating value out of thin air). Similar attacks are traced on the protocol level and should be suppressed by the validators.

If the vast majority of validators collude, then the result could be transaction censorship or even service denial of the entire system. Therefore in the context of the energy conservation law mentioned above, it is important that auditors and asset custodians are also maintaining the full system node and are thus able to **immediately** notice the protocol rules violations and take the necessary actions.

Governance

There is also a number of threats related to the management of the accounting system. Some of them are as follows:

- *An attacker can access equipment of other system participants and thereby gain control over the transaction confirmation process or other participants' accounts*
- *If a large number of validators disagree, consensus cannot be reached (moreover, the rest of the honest nodes will not be able to re-elect validators since this requires the majority of votes)*
- *An attacker can access several validator nodes; as a result, consensus may not be reached*

Since validator nodes are the basis of an accounting system—because they reach consensus regarding the respective database state—their security is of

paramount importance. Validator nodes should be physically distributed, and, if possible, controlled by different organizations (or different structural units of an organization).

Integrity of data

There is also a threat that an administrator, who is responsible for storing data, modifies it without permission. This can be mitigated by placing the hash values of the data in the associated transactions, hence making it possible to verify the data integrity.

Network authenticity

An attacker can claim to be any of the system participants and may even try to mimic an entire network: if an attacker is an intermediary between a particular user and the entire network, he may impersonate various system participants, meaning that while a user thinks that she is communicating with honest nodes, she, in actual fact, is connected to an attacker [20]. Thus, an attacker can impose an alternative history on a particular user or on a group of users (Figure 2.9).

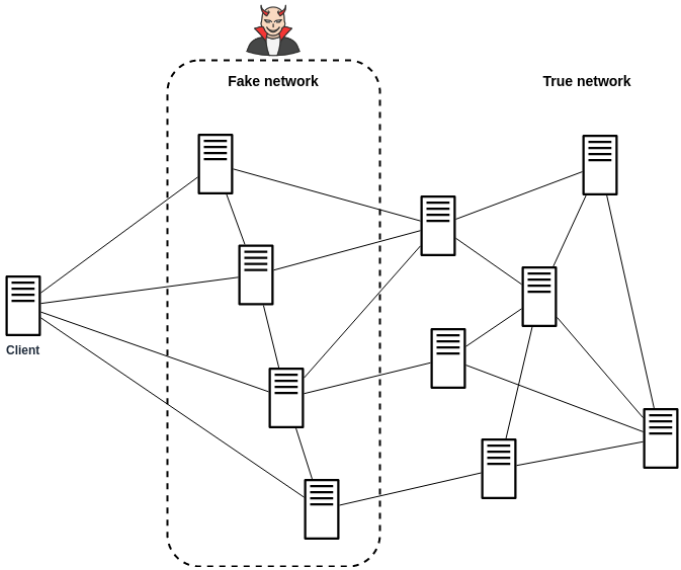

Figure 2.9 — An attacker mimics the network and sends fake data to a client

Accessibility

An attacker can both limit permissions of a particular user to perform available operations as well as grant permissions to the account that should not have such permissions. Hence, user access violations can either happen through illegal restriction of access or illegal granting of permissions.

As a security measure against such attacks, the platform owner has to split the powers for controlling the platform among several administrators and configure their permissions and weights accurately. Thus, he will distribute responsibility for performing key operations among several independent administrators. As a result, a single administrator would not be able to adjust the platform's configuration independently.

Privacy

An attacker can identify transaction participants by analyzing network traffic. In such a case, the starting point for an attack is the communication between a user and an internet provider (under the assumption that the provider captures the user's incoming and outcoming traffic).

Another attack is that an attacker can access users' personal data by hacking the database of an entity that identified and registered the users in its system. In this case, the attacker gains access to all personal data of the user and moreover will determine a connection between the client and their identifier.

Accordingly, the protection mechanism must be that the identity provider complies with the policy of storing and managing personal data, namely that it stores this data and the corresponding key in secret and safely.

Authenticity of decision-making (audit and transparency of the transaction history)

If all the validators of the accounting system are governed by the platform owner, then theoretically, she can force them to roll back the ledger's state and change the list of transactions retroactively (validators will not be able to change the contents of past transactions since they are signed with the users' keys). However, in this case, there must be an auditor who will notice a change in the transaction history and alarm the respective regulatory parties (Figure 2.10).

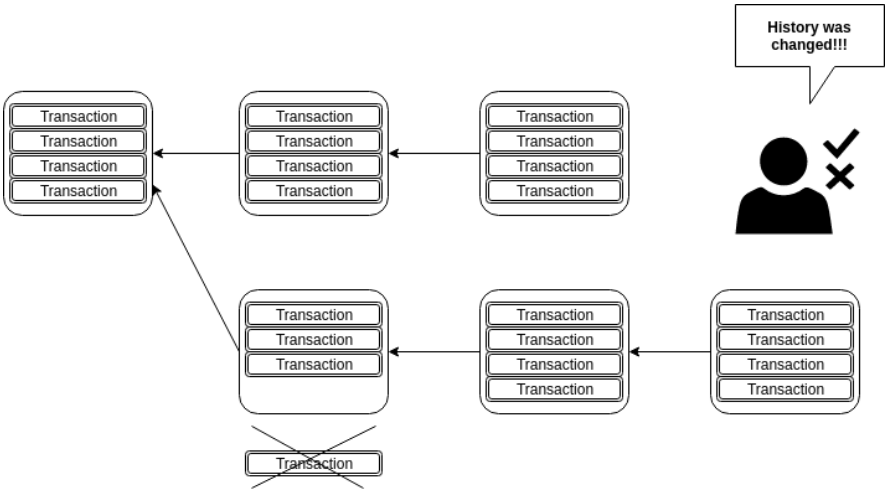

Figure 2.10 — An auditor watches after you

This auditor could be also an ordinary user who keeps all receipts of his own transactions. Also, the user's software stores all the user's requests and the accounting system's responses to these requests. Hence, if the user was outwitted, he will have the proof to show to the governing regulatory body.

Other attacks and defense mechanisms

An attacker can provide the software (or the software update) with a built-in backdoor or another unintended critical vulnerability. Also, an attacker can create fake sites and wallets where inaccurate information about the account states is provided. It is recommended to use open source software (the probability of a backdoor in such a case is much lower) as well as to never rush to update the software immediately after the new update has appeared (it is better to wait for the code to be audited).

An attacker can perform operations which he has no permission to perform. To avoid this threat, platform administrators must strictly regulate the permissions they give to user accounts (the permission sets and relevant processes are described in more detail in section 3).

3 ACCOUNTS

3.1 Accounts and roles

Account is the data structure that contains information about all assets and permissions, and it is controlled by a cryptographic key

Role-based user permission management models grant permission to any given user by associating each user with a particular role and assigning each user a predefined role with a set of permissions. This is opposed to models that grant permissions to users directly. This approach has proven itself at the stage of mainframes, where access to a local machine (and functions performed on it) was limited depending on the user role (admin, guest, etc.). The next step was to use a role-based model to differentiate permissions in huge network systems composed of tens to millions of individual users and network components. Thus, the process of roles allocation has become more complex and demanding, and as the number of users in a system grows, the cost of modifying each user's permissions grows as well. In a role-based model, changing permissions for many users can be achieved in a single transaction due to the fact that the system represents permissions in a one-to-many relationship between any "role" and multiple users [16; 17].

The next step has been the creation of cryptocurrency: a decentralized accounting system where every participant can individually choose a role he wants to perform (e.g., validator, auditor, user, etc.). The model of the role-based permission management has remained in place, but it acquired a completely different, permissionless nature—every participant chooses their role and by default has all permissions to perform it. Although this approach has worked well in cryptocurrency systems, the role allocation mechanism should be more hierarchical and permissioned for the asset management platforms since they are regulated.

Modern asset management platforms require the use of asymmetric cryptography to confirm all user actions within the accounting system. Each change in the state of the ledger must be initiated by the transaction which is signed with the participant's private key (or keys). A transaction is considered

valid only if the user who signed it has permissions to conduct all operations that were placed in the body of the transaction.

This section describes a scheme of how user permissions are assigned based on the role associated with a particular public key. It also describes the structure of an account and the permission distribution between the account signers.

Accounts

An account is a primary unit on the asset management platform. Each operation in the accounting system is associated with the account responsible for the initiation of that operation.

Each participant in the system owns at least one key pair: a public key which acts as an account identifier and a private key which allows the participant to sign operations, thereby proving that they are the initiator of the operation. Each account is associated with a specific role it can perform in the accounting system. All account-related operations—creation, update, deletion, etc.—must also be initiated by transactions and signed by a participant with appropriate permissions.

Primary account fields are as follows (Table 3.1):

Table 3.1 — Account structure

Account ID	Identifier of account
Referrer	Identifier of another account that introduced this account into the system
RoleID	Identifier of the role that will be attached to the account
Balances	Information about the assets belonging to the account and their amount
Signers Data	Array of data about destination account signers to be created (more in 3.2)

Lifecycle of account

Account lifecycle consists of the following steps:

- *Creation*
- *Obtaining permissions*
- *Transacting*
- *Updating*
- *Deleting*

The account's lifecycle is managed by modules such as user wallets (to generate keys, create signatures, restore an account) and admin applications (to create an account, issue and update permissions, etc.).

Roles

The role on the asset management platform is represented by a set of operations permitted for an account. The maximum number of roles on the platform is equal to:

$$Roles\ amount = \sum_{i=1}^{n} \frac{n!}{i!(n-i)!}$$

where n is the total number of operations possible.

The actual number of roles on the platform is defined by its platform owner and depends on the management policy.

Each role in the system has its own identifier. When creating an account, an administrator specifies the role that this account can perform. Since most users in the system have similar permissions (for example, a regular user only has permissions to transfer and exchange assets), the role-based approach is more effective and less expensive than manually setting permissions for each individual account (although this feature is also provided). Accordingly, it is fairly simple to automate the process of distributing user permissions based on the data obtained. As a consequence, the permissions of a user of an accounting system specifically depend on their role in the system (a set of operations available for execution).

Each role is defined by the list of rules. Each rule holds an operation Resource (an entity under which an operation can be performed; e.g., an account or asset), an Action (e.g., creation, transfer, etc.) and the Forbids parameter (a boolean value that indicates whether the specified operation is allowed or prohibited) (Figure 3.1) [18].

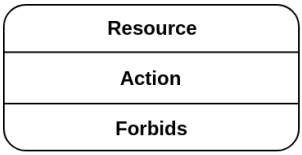

Figure 3.1 — Rule structure

3.2 Signers

Distribution of permissions among several keys is an additional measure to protect access to operations and assets

Signers are structural account objects. Signers are used to allocate all account permissions (which are initially tied to one key pair) among several keys, each containing a smaller set of permissions (Figure 3.2).

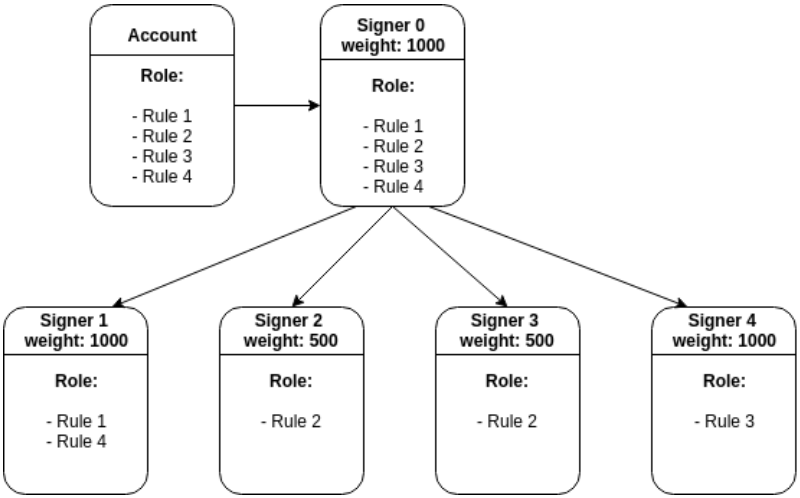

Figure 3.2 — Permission separation process

Each signer, as well as account, has a role. The role of each signer is determined by the account owner and is tied to the signer's public key. Also, each of the signers has a weight, which will be considered when signing a transaction.

When creating an account, one signer is attached to it by default, and its role is identical to the role of an account, and the key weight is maximal. If a user wants to distribute their permissions among several keys, he sends a transaction that initiates the addition of new account signers with specific roles, public keys, and their weights.

The basic data of a signer is presented in Table 3.2:

Table 3.2 — Signer structure

Public Key	Public key of a signer
Role ID	Identifier of the role that will be attached to the signer
Weight	Weight that the signer will have
Identity	If there are some signers with equal identity, only one signer will be chosen (either the one with the biggest weight or the one who was the first to satisfy the threshold)
Details	Arbitrary object with details that will be attached to the signer

Process of permissions allocation

On the asset management platform (AMP) there is initially only one account, the master account, which has permissions to perform all administrative operations on the platform.

The initial roles distribution in the system implies the distribution of system administrator roles; their permissions may differ but should, nevertheless, cover all administrative operations in the system. In order to allocate permissions, the master account signs transactions containing the creation of administrator accounts with the corresponding permissions.

During the platform normal operation, user accounts are created by administrators who have previously been granted the appropriate permissions. User accounts have a set of permissions which were defined by the administrators who initiated the account creation. Changes to account permissions must also be accompanied by an appropriate transaction from an administrator who has the required permissions. After creating an account, the user can distribute his permissions among multiple signers (Figure 3.3).

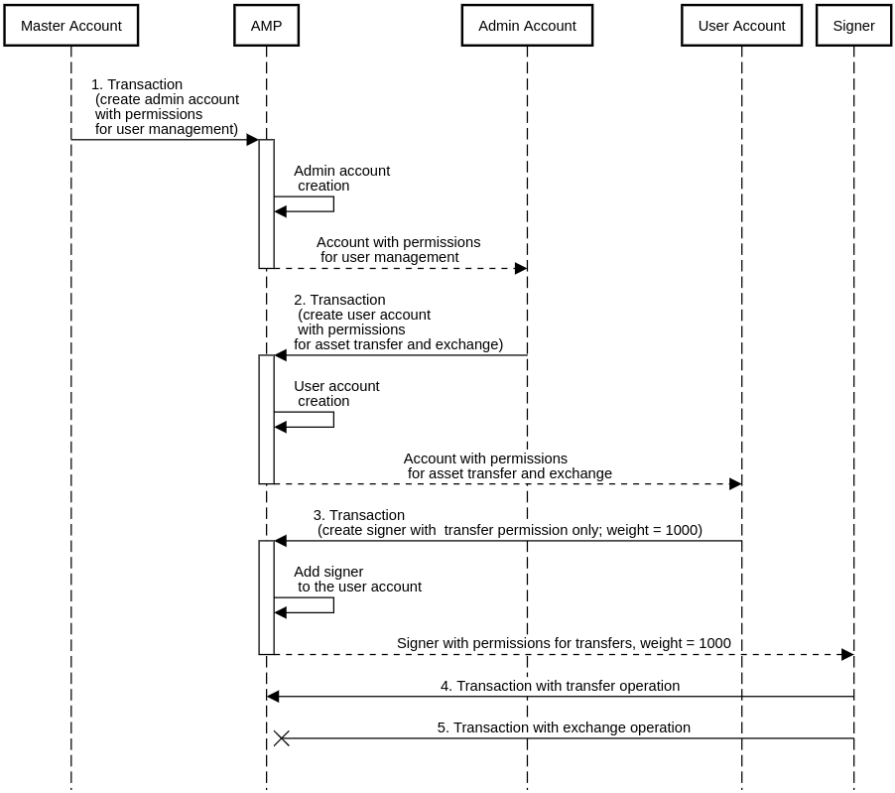

Figure 3.3 — Sequence diagram for organizing the role infrastructure

The process of permissions allocation includes the following steps:

1. Master account sends a transaction to create an administrator account with permissions to manage user accounts (e.g., create accounts,

update, etc.). The asset management platform processes this transaction and creates an administrator account.

2. Administrator sends a transaction to create a user account with permissions to make transfers and then exchanges certain assets. The platform processes the transaction and creates an account with appropriate permissions.
3. User wants to distribute the control of different operations among several keys. She sends a transaction to the platform which indicates the creation of a new signer with the maximum weight and permissions to conduct the transfer operation. The platform processes the transaction and adds it to the account of another signer with permission to conduct transfers.
4. Now, using the signer's private key, a user can sign a transaction containing the transfer operation and send it to the platform. During the transaction verification, the system determines that the key it was signed with has permissions to perform transfer operations, and confirms the transaction.
5. If a transaction that contains an exchange operation is signed with a private key of the same signer, then such a transaction cannot be confirmed since the specified signer does not have the necessary permissions.

Transaction verification mechanism

When publishing a transaction to the network, the platform validators must verify all operations contained in the transaction. For verification, two auxiliary modules are used: *RBAC* (role-based access control) *Account* and *RBAC Signer* (these are additional validator modules). RBAC Account is a module that checks the user's account permissions to perform the operations declared in the transaction. Verification of the user account permissions means verifying that the role of this account has all the necessary rules prescribed. RBAC Signer is a module that checks the permissions of an account's signers in accordance with the declared operations.

The entire transaction verification process can be described using the

following sequence diagram (Figure 3.4).

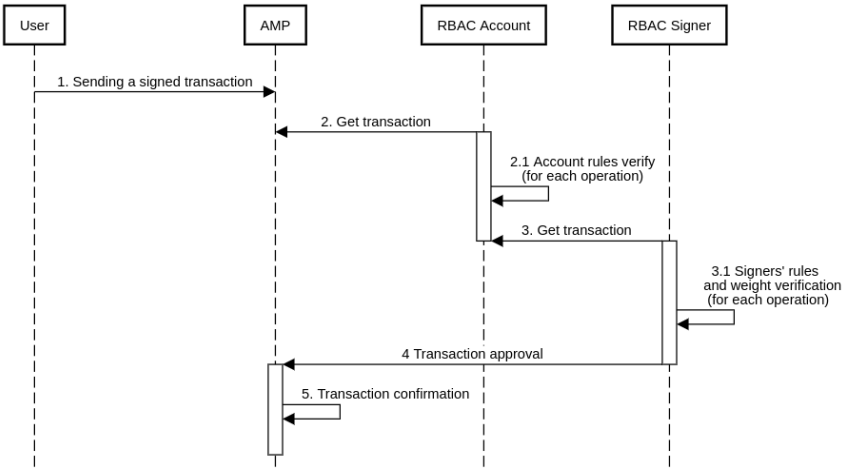

Figure 3.4 — Transaction verification flow

In more detail, this process can be described as follows:

1. User creates a transaction with a set of some operations, signs it, and sends it to the asset management platform.
2. RBAC Account module accepts the transaction from the asset management platform.
   2.1. RBAC Account module verifies that the accounts that participate in each operation have the necessary rules for its execution.
3. Transaction is then sent to the RBAC Signer module.
   3.1. Using the keys with which the transaction was signed, the RBAC Signer module verifies the permissions of the signers for each operation. The signer or group of signers must belong to a certain role in the system and have a total key weight equal to or higher than the weight threshold to perform an operation.
4. If the transaction was verified and the signer actually has the corresponding permissions, the RBAC signer module informs the asset management platform that the transaction is valid and the operations specified in it can be performed.

5. The transaction is validated and the state of the ledger changes in accordance with the specified operations.

Example of how a transaction containing a payment operation is verified

The following sequence diagram shows an example of how a payment operation which involves two accounts (sender and receiver) is verified. In order for a transaction containing this operation to be successful, the following conditions must be satisfied (not including checking fees, limits, etc.):

1. Sender account must feature a rule that allows transferring a particular asset.
2. Sender account's signer must feature a rule that allows transferring a particular asset.
3. Recipient account must feature a rule that allows her to accept transfers in the relevant asset.

The payment transaction verification process can be described using the following sequence diagram (Figure 3.5).

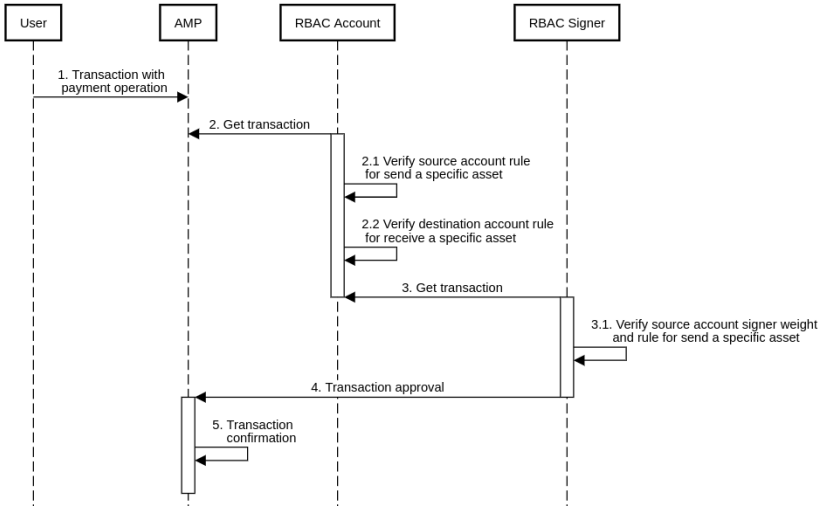

Figure 3.5 — The process of confirming the transaction containing a payment operation

In more detail, this process can be described as follows:

1. User sends a transaction with a payment operation that transfers some amount of an asset to the balance of the recipient account.
2. RBAC Account module accepts the transaction.
   2.1. RBAC Account verifies that the sender account has permission to transfer a particular asset.
   2.2. RBAC Account verifies that the recipient account has permission to receive a particular asset.
3. If the RBAC Account verifications are successful, then the RBAC Signer accepts the transaction.
   3.1. RBAC Signer verifies that the sender account's signer has permission to transfer a particular asset as well as that the total signer weight is sufficient to perform the declared operations.
4. If the check is successful, the RBAC Signer confirms that the transaction is valid and gives permission for its confirmation.
5. Finally, the transaction is confirmed and the ledger state is updated.

4 ASSETS

4.1 Assets and asset ownership rights

Fractional ownership of assets is one of the key benefits of digitization

A *digital asset* is a digital representation of ownership of a particular asset. It is important that an asset or a fraction of it must necessarily have only one owner at any particular moment in time. An owner of an asset is a user who holds the corresponding tokenized asset (token) on her balance.

Asset data structure

Regardless of the type of an asset and the platform on which it is managed, it is necessary to select the data set that should be tied to it.

- *Code*
- *Maximum issuance amount*
- *Creator and issuer details*
- *Type (policies)*

Each asset type must have its own unique code to distinguish it from all other assets accounted for in the system. Each account in the system has a separate balance for each asset it owns (more specifically, for an asset which an owner of the account has permission to manage).

Each asset type on the platform must also specify the maximum issuance threshold (i.e., the maximum amount value of this asset that can ever be issued). Note, the overall amount of tokenized assets on the platform doesn't increase constantly; assets may also be destroyed (or more properly, redeemed) during the withdrawal. If needed, the *maximum issuance threshold* value can be updated by a separate operation with the approval of an administrator (or administrators) with appropriate permissions.

Every asset must contain the information about its creator (namely, the creator's identifier) as well as identifiers of accounts that can issue this asset.

Thus, each request for the tokenized asset issuance on the platform is verified in terms of whether the keys that signed the request have permissions to issue this asset. Note, the process of asset issuance can be distributed among several parties to diversify risks related to the loss of keys or censorship of decision making.

The last important fragment of the asset data is its type. The type determines the management policy for an asset: which identities can own the asset which permissions must be granted, and whether an asset can be withdrawn from or exchanged on the platform, etc.

Asset ownership rights

Ownership rights for a particular asset are proved by providing an identifier and a pair of keys bound to it. Any action for changing the asset owner must be cryptographically confirmed (i.e., must contain the digital signature of the previous asset owner). This allows the reliable transfer of ownership rights for an asset without the need to transfer the asset itself. The accounting system simply rewrites balances, while the custodian stores the actual asset (e.g., the physical bar of gold, private keys from bitcoin addresses, etc.). The custodian must monitor that the total asset amount on the platform matches the actual asset amount.

To obtain the actual assets from the custodian, a client must contact her and prove that he owns a particular amount of asset. After that, the asset is withdrawn from the platform. This operation involves the redemption of assets on the platform—a client cannot simultaneously hold both the actual asset and the digitized ownership right for this asset.

4.2 Asset management processes

The essence of secure and convenient asset management lies in the use of cryptographic keys

Tokenized asset lifecycle

Every tokenized asset has its own lifecycle. The basic stages are as follows:

- *Creation*
- *Pre-issuance*
- *Issuance and distribution*
- *Transfer*
- *Trade (exchange)*
- *Redemption*

Below, we will consider the features of each stage.

Creation, pre-issuance, and issuance

During the creation stage, a tokenized asset cannot be traded or transferred because at this point it doesn't actually exist. This is simply an announcement of the tokenized asset details: name, max issuance amount, permissions required to interact with this asset, etc.

Pre-issuance is the stage where an asset has been created but is not yet in circulation. After the pre-issuance stage, actual tokenized assets already exist, but no one yet owns them—they are all locked up in a smart contract. Pre-issuance is either done automatically or manually. Automatic pre-issuance means that a certain amount of asset specified during the creation will be pre-issued immediately after the administrator approves the creation request. Manual pre-issuance means nothing happens after the creation request approval, and the issuer has to manually create a pre-issuance file (for higher security, this could be done via the offline application), then uploads it on the platform, and waits for the administrator's approval.

During the issuance stage, assets are being put into circulation. This process can be performed either through the direct issuance (manually or through the PSIM, payment system integration module) or through the crowdfunding campaign. Direct issuance presumes a manual transfer of the asset to user balances. A crowdfunding campaign can be created by any user with corresponding permissions and should be approved by the administrator (only the tokenized asset creator is able to put the asset up for sale). More details about issuance management will be provided in section 4.3.

Transfer

Unlike the traditional order execution model, where users send requests to the main server to transfer funds and the server responds by rewriting the balances correspondingly, the asset management platform supports direct transfers between users. In particular, all transactions are initiated by the users themselves, using cryptographic keys.

Note that for a tokenized asset to be transferred, both the sender and the receiver need to have corresponding permissions. Otherwise, the system will return an error, and the transaction won't be submitted.

The FBA consensus predominantly implemented in asset management platforms allows achieving full finality of transactions [19]. This means that a transaction is irreversible; once it has been accepted by the validators and recorded on the ledger, the transaction cannot be canceled. We will observe how the FBA consensus works in section 5.3.

Trade

An asset management platform needs to support an internal exchange module that allows its users to exchange assets issued on the platform without the need to integrate with external exchange services. Transfers of assets between clients of different accounting systems are implemented using atomic swaps.

Redemption

Tokenized asset redemption is the process of taking a corresponding amount of asset out of circulation. One example of the redemption process is the withdrawal process.

4.3 Issuance management

Secure asset management lifecycle is necessary to digitize assets

The issuance of an asset on the tokenization platform is a procedure that requires the highest security levels. Issuance and management of assets is a

crucial task and the biggest difference from the traditional asset management is that now it becomes decentralized (through multisignature during asset issuance). The issuance is divided into three stages: creation, pre-issuance, and issuance. In this section, we will examine the features of each stage as well as the security issues related to them.

- *Creation*
- *Pre-issuance*
- *Issuance*

Tokenized asset creation flow

As noted earlier, the creation of an asset does not imply its issuance. During this process, a user determines the name of an asset, its properties (deposit / withdrawal possibility, etc.) as well as the requirements regarding its regulation (permission which an account must have to operate with this asset). The process of creating a tokenized asset is shown in Figure 4.1.

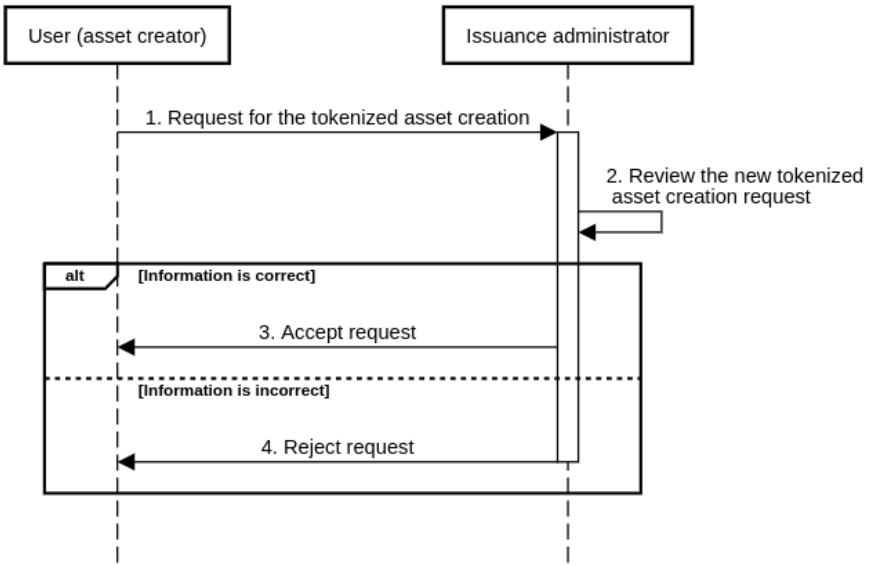

Figure 4.1 — Asset creation flow

The process of creating a tokenized asset is as follows:

1. User creates a request for asset creation. The request includes all the required information for releasing the asset. The request is sent to the platform administrator who is responsible for the issuance.
2. Administrator receives the request and processes it.
3. If the provided information is sufficient and correct, the administrator confirms the request, and the corresponding asset appears on the platform.
4. If the provided information is not sufficient or not valid, the administrator rejects the request. In the refusal, the reasons of why the platform administrator rejected a specific request must be specified.

Tokenized asset pre-issuance flow

The pre-issuance is required to create the required amount of tokenized assets on the platform (Figure 4.2). In case of successful pre-issuance, tokenized assets can be distributed among the users (manually or as a result of a crowdfunding campaign).

The process of pre-issuing a tokenized asset is as follows:

1. User creates a pre-issuance file with all the necessary details. It is possible to set a condition requiring several signatures to perform pre-issuance. In this way, the process and, correspondingly, the responsibility of managing asset pre-issuance can be shared between several parties. At this stage, using an offline application is preferable for security purposes.
2. User signs and saves the file using an offline application.
3. User (it could either be the same user or another responsible party) downloads the pre-issuance file and verifies the details specified in it.
4. If he agrees with all the details, he signs it with his private key and creates the pre-issuance request.
5. The request is sent to the platform administrator for further processing.

6. The platform administrator (the one responsible for the issuance) reviews the request details and verifies the necessary signatures.

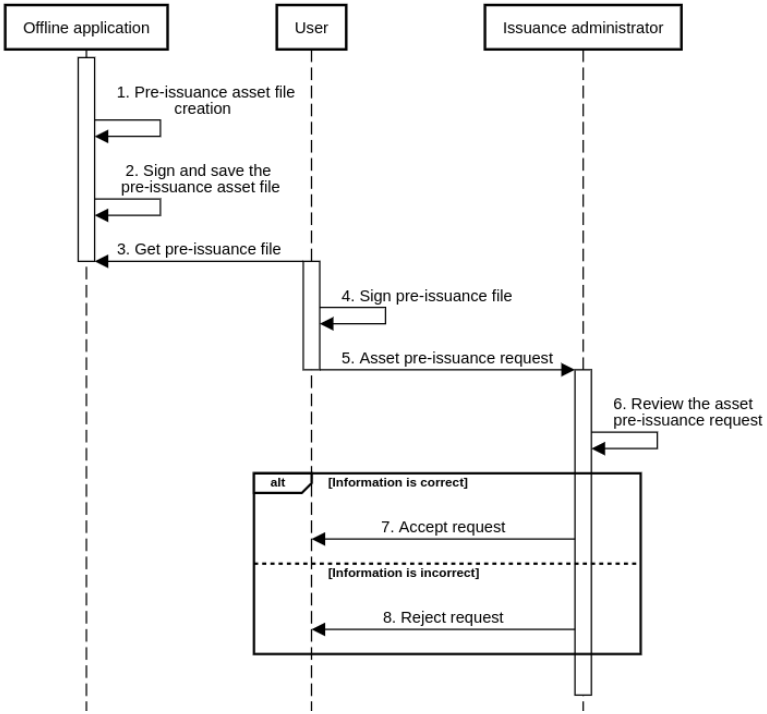

Figure 4.2 — Asset pre-issuance flow

7. If the administrator agrees with the pre-issuance details, she confirms the request.
8. If not, the administrator rejects the request and specifies the rejection reasons.

Tokenized asset issuance flow

After the tokenized asset pre-issuance is successfully completed, an asset can be issued, that is, distributed among the platform participants (Figure 4.3). The way it is distributed is decided by the asset owner. It may simply be the distribution of equal amounts of a tokenized asset to all platform participants

with appropriate permissions (for example, one of the possible e-voting implementations). Alternatively, a user can carry out a crowdfunding campaign (for further details, see 9.2), and a tokenized asset will be distributed according to the investors' shares.

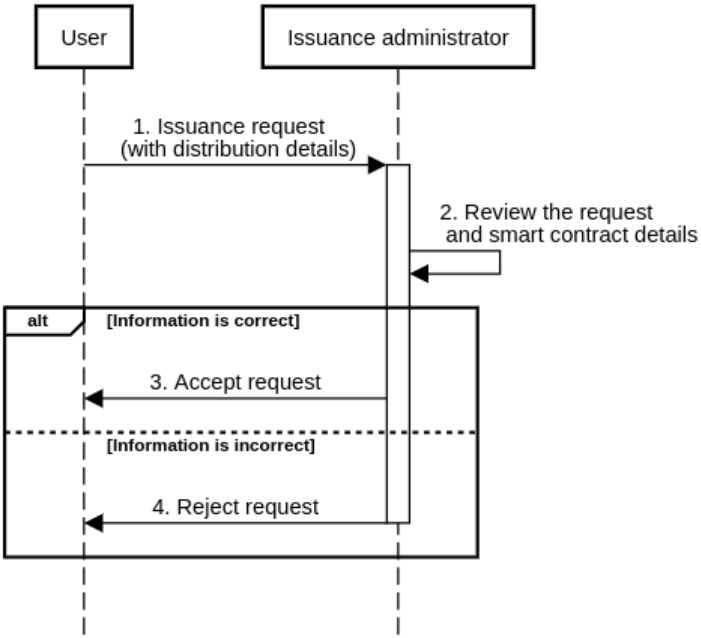

Figure 4.3 — Asset issuance flow

The process of tokenized asset issuance is as follows:

1. User submits the issuance request with all the necessary details (the predetermined distribution or a crowdfunding campaign distribution).
2. Administrator analyzes the request details.
3. If the provided information is correct, the administrator confirms the issuance request.
4. If the provided information is not correct, the administrator rejects the request (the rejection reasons must be specified).

4.4 Asset transfer and trading. Fees and limits

The management of transactions, fees, and limits can also be carried out using cryptographic keys

Transfers

Every transfer within an accounting system must be initiated by a transaction that contains the corresponding operation. The basic data in the transfer operation includes the following:

- *Source account identifier (i.e., who sends an asset)*
- *Destination account identifier (i.e., who receives an asset)*
- *Amount of asset sent*
- *Fee data (for both the sender and the receiver)*
- *Subject (payment purpose)*

After a transaction is signed, it is sent to validators for further verification and confirmation. We now discuss in detail how the provided data is verified.

First, the account specified in the operation is inspected to confirm it holds the necessary amount of asset for transfer. Once it is confirmed that the account's balance is sufficient, it is verified that the specified fee is sufficient (the fee can be set for both the payment sender and the recipient).

In the next step, the counterparts' permissions are verified, i.e., that the sender has the necessary permission to transfer the specified amount of the asset underlying the transaction and that the recipient is permissioned to receive it. The verification is carried out according to the sender's account requirements and the keys with which the transaction is signed (for further details about signers, see section 3.2). If all the verifications above are successful, then the transaction is confirmed by validators.

Notably, the module for checking account permissions in terms of asset management is auxiliary and doesn't relate to accounting system's core. However, the transaction must be signed by this module even before it is validated on the platform—the accounting system must be sure that the module responsible for checking permissions have validated the transaction (it's not that

important for the system how specifically the verification was carried out; it just cares about the result).

Trading on the internal exchange

Accounts in the system can initiate asset exchange operations. The most crucial here is that assets exchanged were previously issued on the platform. Any user account (with appropriate permissions) can view orders for purchasing and selling assets. If the user is satisfied with an existing order (namely with the buy/sell conditions), she can participate in it; otherwise, she can create a new order and specify her own details, such as a different price. Note that exchange on the platform is an atomic operation, meaning that intermediaries are excluded, and the participants are guaranteed to receive each others' assets according to the specified conditions.

The principles of atomic swap and multisignature mechanisms are described in more detail in sections 10.2 and 10.1 respectively.

Fees

Any asset management platform has to provide advanced tools to set and configure fees. The responsibility for fee management lies with the specific administrator, provided he has the appropriate permissions. The fees can be set and changed according to one or more of the following parameters:

- *Account*
- *Asset*
- *Operation*

The platform must support a flexible adjustment of fees that match the requirements of a specific account or a group of accounts. Now, fees can be configured based on, for example, the geographic location, the target client or the regulatory body associated with the account. Also, in addition to being able to impose a particular transaction with fees, it is important that there is an ability to impose a particular asset. Hence, each transaction transferring a particular asset will be imposed with the set fee (it could be the minimal amount or even zero). However, when a client wants to withdraw, then she will very likely need

to pay fees—this could be due to, for example, expenses owed to the custodian or other parties involved—and this ability should be provided for the platform owner.

The fee can be set either as a constant value or as a percentage of the transaction amount. This parameter can be adjusted depending on the account type, asset type (a particular asset that participates in the operation and with which the fee is paid), and the particular operation.

Limits

The responsibility for managing limits lies with a particular administrator or a group of administrators. Limits can be set for specific assets specifically configured for transfer, withdrawal, exchange, etc. Also daily, weekly, and monthly limits can be set.

Similar to fees, limits can be set for a particular account, group of accounts, or an account type depending on the operation, a set of operations, or a specific asset. Setting, updating, and removing limits must also be followed by a corresponding transaction.

Limits verification is performed by a separate module at the stage of pre-validation. The accounting system only needs to receive a signed confirmation from the appropriate administrator that the transaction has passed the verification (while it may not even matter how specifically the verification was conducted). All transactions associated with the account are verified during the specified period. If the total amount of transfers of a particular asset is within the permissible limits, the transaction can be confirmed.

4.5 Mapping physical assets to the digital infrastructure

Introduction of new security standards into banking applications via cryptographic techniques

Cryptocurrencies have introduced a new way for maintaining the digital financial infrastructure—the one which presumes peer-to-peer transfer of ownership between users (via cryptographic signatures), lower expenses on the infrastructure maintenance as well as new security standards (no physical

protection of servers and no firewalls).

However, for obvious reasons, such as the inability of regulation and high price volatility, cryptocurrencies turn out to be ineffective in the business domain.

This gave birth to a new trend: digital assets backed by those with a stable price (e.g., dollars, gold, etc.), which can be used for transactions in financial organizations. It is noteworthy, however, that the breakthrough component lies not in the stability of the price, which is solely the feature of an asset that is being tokenized. It is rather the technological innovation of the tokenization, which takes the best features from cryptocurrencies and combines them with new features. For the first time it is possible to implement digital assets within the regulatory frameworks and business environments, where all parties to a transaction have to be identified, all risks associated with the transaction need to be minimized, and actions should be traceable and, if needed, restricted.

Principles of the currencies tokenization

This scheme presumes accounts which store funds of users in the payment system. The total amount of assets should be equal to the amount of money stored on the accounts.

NOTE

In fact, this scheme is actually regulated by an e-money license and works in many countries (an example of such a system is PayPal [21]).

NOTE

If the case is about a major bank with a large number of branches, then it makes sense to establish an infrastructure for this bank that presumes each branch has a full network node synchronizing in real time with the other branches. This eliminates a single point of failure (if hackers manage to break into the servers of a local bank's branch and try to rewrite balances, this will by no means affect the overall system's state) and significantly increases the responsiveness of the bank since it distinctly knows how much actual money each branch has since they are all synchronized in real time.

When a particular user makes a request that she wants to deposit money

from some payment system (such as PayPal) to the bank's accounting system, it is submitted to the user account in the bank's accounting system (Figure 4.4).

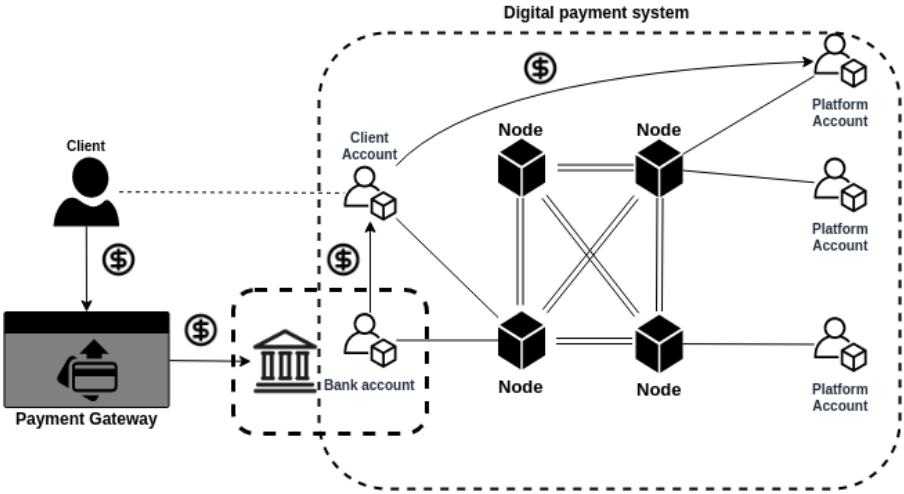

Figure 4.4 — Using accounts for deposits and transfers

What actually happens at this moment is that the payment system sends a message to the accounting system using API, and the corresponding amount of assets is issued to the bank's client account. The next step could be the transfer of funds between users of the bank (Figure 4.5).

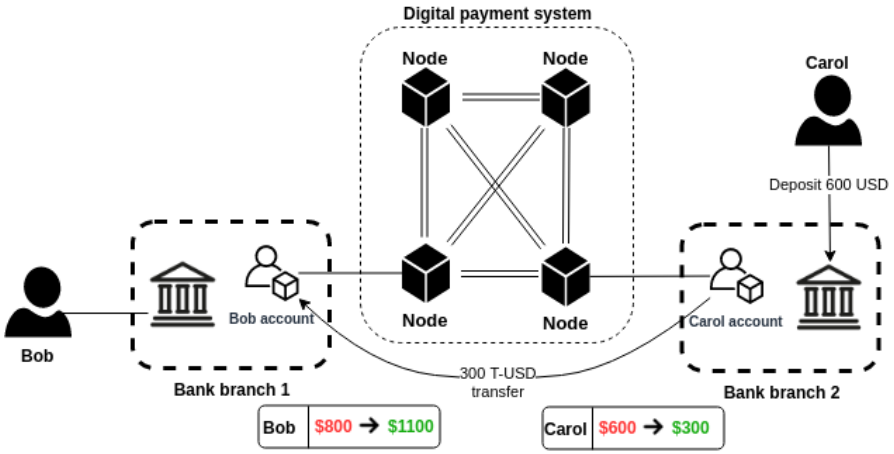

Figure 4.5 — Transfer between users

When it comes to sending money from one user to another, e.g. Carol who is a user of bank branch 2 sends money to Bob who is a user of bank branch 1, then the benefit becomes evident: since both branch 1 and 2 maintain the system nodes with the full transactions history synchronized in real time, the transfer process for the bank's accounting system becomes seamless, real-time, and cost-effective. The next step could be the withdrawal of money from the accounting system (Figure 4.6).

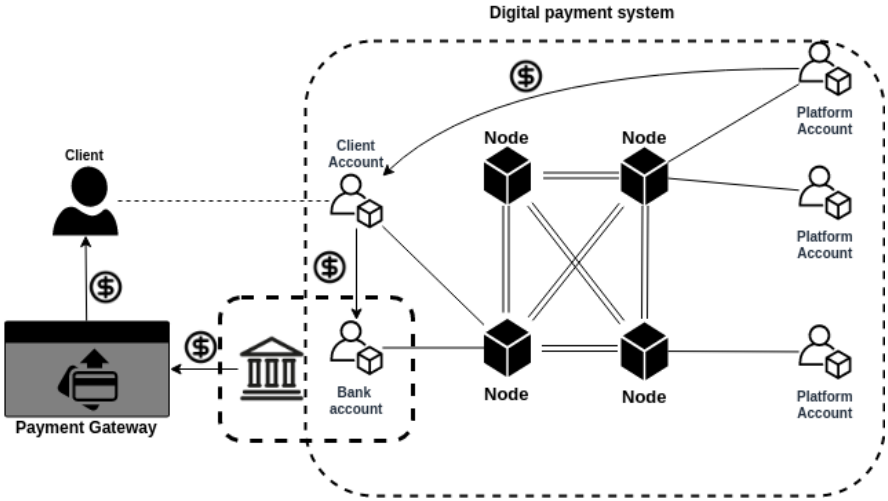

Figure 4.6 — Using a concentration bank account for withdrawal

When a user creates a withdrawal request, the corresponding tokenized assets are transferred to the bank account, and then the bank transfers funds to the user account in the external payment system (or, for example, dispenses cash to the client and destroys the corresponding tokenized assets).

5 TRANSACTIONS

5.1 Transaction structure and lifecycle

Each transaction is always signed

Each transaction is processed individually

Each transaction is associated with the initiator account(s)

Transactions on the asset management platform are different from transactions in traditional financial systems: as opposed to the traditional approach, where user requests are processed within a single authentication session, every transaction on the asset management platform passes an independent validation procedure; every transaction is verified depending on the permissions granted to the corresponding public key.

Unlike cryptocurrencies, which do not make any requirements for the user identification, every transaction on the asset management platform is initiated by a specific account, and every account is bound to a specific identity. Moreover, compared to cryptocurrencies, an asset management platform supports a significantly larger number of possible asset operations, and the number of possible operations can be extended quite easily.

In this section, we will analyze how transactions are configured on the asset management platform. Further, we describe how operations are performed and verified as well as provide definitions of accounts, permissions, and roles, and how they are related to users‘ rights to perform operations.

Transaction structure

Each change (state update) in the ledger must be initiated by a transaction. A transaction contains a set of operations, information about who is the transaction initiator, and a set of signatures that are necessary for its application. The transaction is *atomic*. This means that if at least one operation in the transaction cannot be performed, then the entire transaction is rejected. The transaction structure is shown in Table 5.1.

Table 5.1 — Transaction structure

Source account	Account used to run the transaction
Salt	Random number used to ensure there are no hash collisions
TimeBounds	Validity range (inclusive) for the last ledger close time
Memo	Allows attaching additional data to transactions
Operations	List of operations to be applied
Signatures	List of signatures used to authorize a transaction

So why does a transaction contain more than one operation in it? The answer is, to provide security and atomicity. Some actions, such as the exchange of assets, must be carried out atomically without any intermediary. The proposed structure allows including two operations (one for the transfer of assets in one direction, and the other in the opposite direction) in a single transaction. This means that the situation when funds are received by only one of the two parties is simply excluded—operations are either performed both or not performed at all.

Operation structure

Operations are commands to change the ledger state. Note that an asset management platform usually supports a large number of operations. For each operation, it must be verified that an initiating account has the permission to perform it. Also, each operation has a weight threshold, which is determined by the protocol rules. When verifying an operation, one needs to verify:

- Whether the initiating account has permission to execute it
- Whether the total weight of the keys (account signers, for further details, see section 3) used to sign the transaction is covered by the specified operation weight threshold

The structure of an operation is as follows (Table 5.2):

Table 5.2 — Operation structure

Source account	Account used to run the operation
Body	Operation body

Transaction lifecycle

As noted earlier, a transaction is a set of digital data that initiates certain operations in the system, and its execution changes the state of the ledger.

- *Creation*
- *Signing*
- *Submission*
- *Propagation*
- *Verification*
- *Transaction set formation*
- *Accepting (rejecting)*

Further, we will cover features of each stage of the transaction lifecycle (Figure 5.1) as well as features of the transaction verification at these stages.

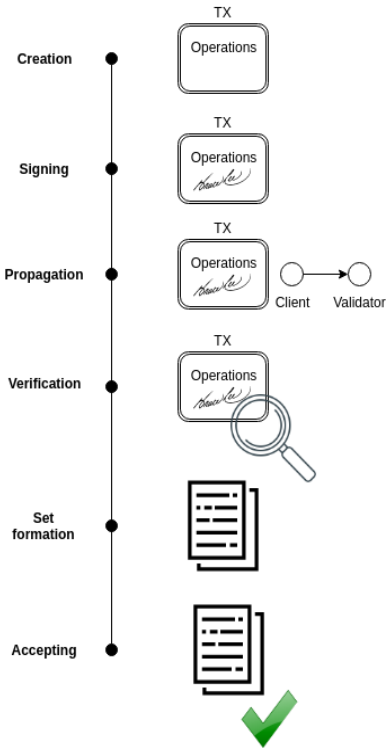

Figure 5.1 — Transaction lifecycle

Transaction creation

The first step in the transaction lifecycle is its creation. A client forms a transaction, specifies his account identifier, generates the salt value, and sets the time frame for it to be accepted.

The salt value is necessary to add additional entropy to the process of computing the transaction identifier. Since one account can initiate several transactions whose fields are identical (for example, to make several identical micropayments), hash values (identifiers) of these transactions will be the same as well. However, it should be noted that only one of the transactions with the same identifier can be confirmed (the rest are considered conflicting and will be rejected). Therefore, when generating a random salt, the probability of creating two transactions with the same hash value tends to zero.

The transaction initiator also determines the time value after which the transaction can be accepted as well as the time value until which it can be accepted. Such a mechanism allows determining the time frame for accepting a transaction: it can be accepted by validators only within a specified time range.

After filling in the listed fields, a user adds the operation (or operations) to be performed with the transaction. In section 3.2, we will describe how user permissions for performing operations on the platform are verified.

Signing

After a user has created a transaction, she must sign it. A digital signature is required for the further authorization (verification and confirmation) of the transaction. Every transaction requires at least one public key to authorize it (the rights for performing operations included in the transaction are verified according to the provided public keys).

In some cases, a transaction may have several signatures. Such situations arise if several accounts participate in performing an operation (for example, an exchange operation) or if an account has several public keys with different weights and / or permissions (for more details, see section 3). To implement such a mechanism, multisignatures and threshold signatures are used [22].

In threshold signatures, every default operation has a weight value that is required for it to be executed. This value is determined in the accounting system protocol. The threshold weight value may differ for particular operations (depending on the protocol rules) and is related to the required security level for a specific operation (for example, micropayments require low weight, while operations to create account signers require high weight; definition of signers is provided in section 3.2.

Submission, propagation, and verification

After the transaction has been created, the user must send it to the validators for confirmation. As soon as a validator receives the transaction from the user (or from another validator), he performs the initial verification: whether the transaction is properly created, whether the public key weights are enough to perform the specified operations, whether the fee is enough to conduct the transaction, etc. If the transaction has passed the initial verification, it is sent to all other validators via the network connection. Finally, this transaction is propagated among all validator nodes.

> *NOTE*
>
> *There can be only one validator. In this case, he solely verifies and confirms all transactions.*

Transaction set formation

At each defined time period, a validator node creates a set of transactions that have been received since the last reconciliation of the ledger state. As a result, each validator has its own set of transactions at any particular moment in time. Next, nodes reach consensus on the set of transactions that must be confirmed. The process of reaching consensus will be described in detail in section 5.3.

Accepting transactions and changing the ledger state

After consensus is reached, the set of transactions reconciled by the validator nodes changes the ledger state. If one of the transaction operations cannot be performed, the transaction is rejected entirely. After reconciling and accepting all new transactions, validator nodes form a new ledger state, the one which all honest nodes agree with. Hence a fee is required to be paid for the operations performed, and it is withdrawn from the balances of accounts that initiated the confirmed transactions. This process of new transactions validation and ledger state update then repeats itself.

5.2 Transaction storage principles

Transactions are stored in a linked form that ensures the integrity of their history

Transaction storage

The primary ledger of the platform implements a linked structure of transaction sets. Every new transaction set defines a new state of the core in relation to the previous state. Herein, the ledger's integrity is ensured through links between the transaction sets (each set is cryptographically linked to the previous set). Going forward, this enables an ongoing real-time validation of the database and the history of all transactions contained herein. The primary database stores all the data passing through the core.

How transaction sets are linked to each other

Each transaction set contains a link to the previous set. A unique identifier (a hash of the previous transaction set) is used as a link (Figure 5.2). If one of the transaction sets is attempted to be modified, all further sets must also be modified (since they can no longer link to the previous transaction sets) [23].

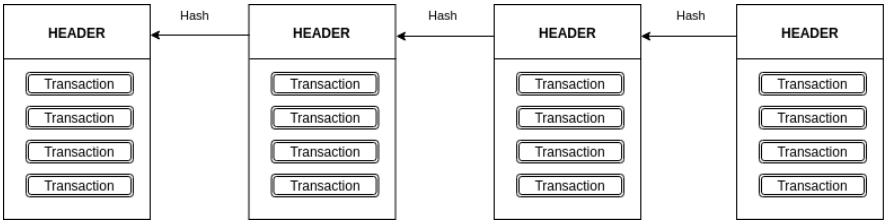

Figure 5.2 — Linked sets of transactions

Modifying at least one transaction from the past would lead to a chain reaction: all further data sets will eventually be modified (Figure 5.3). This ensures the integrity of the ledger data.

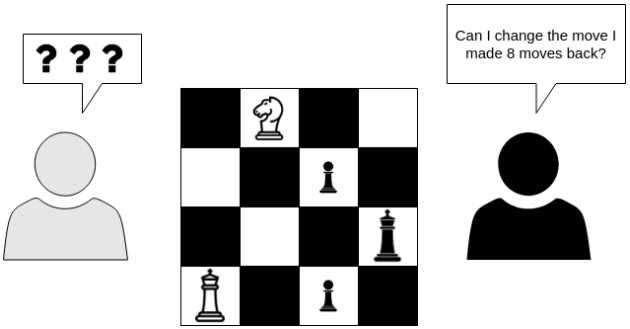

Figure 5.3 — Sometimes to change your previous decision, you need to cancel all the actions following it

How transactions are linked to each other

Some accounting systems imply the direct linking of transactions. An example is the UTXOs accounting model (e.g., Bitcoin, Monero). In such systems, each transaction input explicitly links to the previous transaction output spent. Thus, the ledger state contains a set of unspent outputs to which further transactions will link. These outputs must be unlocked using the specific private keys.

Another way is to use accounts and balances. In this case, a specific transaction must be initiated by an appropriate account. Every transaction changes the state of its associated accounts and/or their balances. Note that the final state of an account is reached by performing all operations that are associated with it. Thus, the system participants who can view the entire transaction history can verify that the current state of an account and its balance is indeed reached after the associated transaction has been confirmed.

Updating the accounting system state

The ledger state is reached through the coordination between the validator nodes. If a system is centralized, it has only one validator who personally makes all decisions about updating the ledger state. If, however, the system presumes several validators (business requirements), then they need to reach consensus regarding the database state. Hence, the final system state is considered correct

and the system will continue its operation based on this state only if validators have reached consensus regarding it.

Each platform validator must store the entire history of transaction sets. Keeping this history ordered (transaction sets are linked to each other with a hash) allows validators to restore the relevant ledger state in case of errors.

The ledger state and all transaction sets can be both accessed by everyone (if the system is open) or only by the owner and the validators of a particular platform. Clients can contact validators with the signed requests and receive the related transactions and the current state of balances. Whilst, if the system is public, then each participant can run a full node and receive the necessary data independently.

Forming the final state of an accounting system

In addition to the primary database, there is another database that only stores the final state of the accounting system. It is optimized to read and record the accounting data during the validation of new transactions fast. Thus, each node in the network stores and processes the state of the core using the two databases simultaneously: one for the fast searching and reading of the transaction data, and the other for the support of the general history and for the synchronization with the other nodes in the network.

5.3 Approaches to reaching consensus

Owners of accounting systems (and consortiums) should be free to choose a consensus mechanism between them

- *Multisignature*
- *Proof-of-work*
- *Proof-of-stake*
- *Byzantine fault tolerance*
- *Federated byzantine agreement*

Using multisignature

In terms of its implementation, the simplest method of reaching consensus is using multisignature. In this case, in order to confirm the ledger state update, authorized parties (validators) must sign it. A protocol can prescribe the *n*-of-*m* multisignature scheme, where *m* is the total number of platform validators, and *n* is the number of validators sufficient to update the state.

When a validator creates a new list of transactions, she signs it (thereby claiming that she agrees with this update) and distributes it among the other validators. If the other validators agree with the proposed state (set of transactions), then they sign the proposed block. This procedure is called voting. Once the voting process is completed and the application has gained the required number of signatures, it is considered valid and the ledger state is updated.

Note that in the described approach, validators are public (not anonymous), that is, and everyone can check all others' votes. If a node acts dishonestly (i.e., it votes for invalid decisions, which contradicts the protocol rules, or it refuses to vote for a long time), the other nodes can exclude it from the accepted set of validators.

Proof-of-work and proof-of-stake

Proof-of-work is the first consensus algorithm applied in practice [23]. This method is required for anonymous, permissionless systems. Proof-of-work presumes that validators are anonymous and solve a resource-intensive task. The fact of solving the task is easily verifiable for everyone. For this, the hash function preimage is most often used.

In this case, validators compete with each other. The probability to solve the task first depends on the random value, while the more computational power a network participant has, the higher is his probability to solve the task first. The key advantage of this approach is a high level of decentralization of a system. Virtually every participant can independently verify transactions; there is no need to get permissions.

Proof-of-stake is a similar approach, but in this case, it doesn't require validators to have a certain amount of computational power but rather hold a

stake of native currency coins, shares or other assets on their accounts (more specifically, on their addresses) [24]. In order to confirm a block, validators vote using their own stake (similar to shareholders voting). Compared to proof-of-work, this algorithm is more efficient in terms of system capacity.

Notably, it is important to take into account the cost of attacks on the accounting system. In the proof-of-work case, an attack on the protocol isn't feasible since the costs of the attack implementation would be much higher than the resulting benefit. In the case of proof-of-stake, if validators try to confirm invalid transactions, they risk losing their stakes.

Byzantine fault tolerance

The system based on the *Byzantine fault tolerance* (BFT) algorithm operates with an upfront predetermined set of validators [25]. These validators are known, and they verify each other.

When confirming the next set of transactions, validators share signed messages containing transactions for confirmation with each other. At the beginning of each new stage, a new leader is selected who initially proposes a set of transactions that she considers valid. The following iterations present the exchange of messages between validators (Figure 5.4).

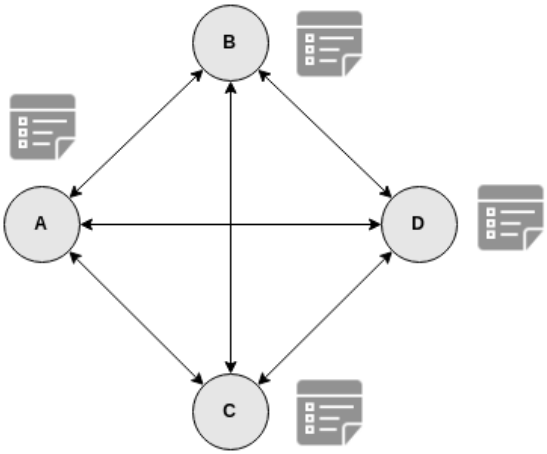

Figure 5.4 — The BFT approach

Finally, after the messages are exchanged with all the participants, each validator creates a set of values for which the majority of nodes has voted. If the number of honest system validator nodes is not less than ⅔, then validators can reach consensus and make sure that their ledger state matches.

The BFT consensus algorithm can be used for a relatively small number of validators, several dozen for example (although, for example, Facebook's Libra claims to have up to 500 validators [26]). This algorithm is distinguished by high capacity compared to previous approaches. It is more centralized and doesn't suit for permissionless environments but rather for the business needs.

Federated byzantine agreement

The primary idea of this type of consensus lies in the intersection of groups whose members reach agreement only with each other, but since participants can simultaneously belong to several separate groups, consensus is reached in the entire network [19]. FBA gets rid of the scalability problems, which are inherent to BFT, by allowing new nodes to join the system while it is already operating. However, only the nodes trusted by others can become validators.

Initially, each network member selects a set of nodes that he will trust. These sets of nodes are called *quorum slices* (Figure 5.5). Selected participants can directly influence the validator in the context of decision-making (as in the BFT protocol, participants influence each other).

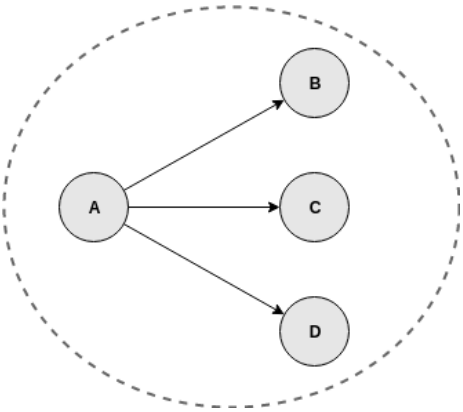

Figure 5.5 — A quorum slice for A

Since the selected nodes (participants of the validator's quorum slice) have their own quorum slices, a quorum is actually built by all nodes included in these slices (Figure 5.6). Consensus is reached among the quorum participants.

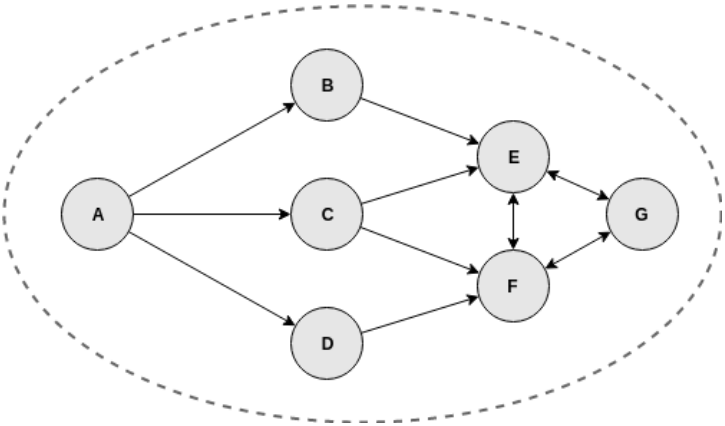

Figure 5.6 — A quorum

Note that there must be intersections between quorum slices (Figure 5.7). If there are no intersections, then the network will be divided into several subnets, each maintaining its own transaction history. In fact, the entire scheme of this consensus mechanism is based on the fact that participants with intersections cannot make conflicting decisions.

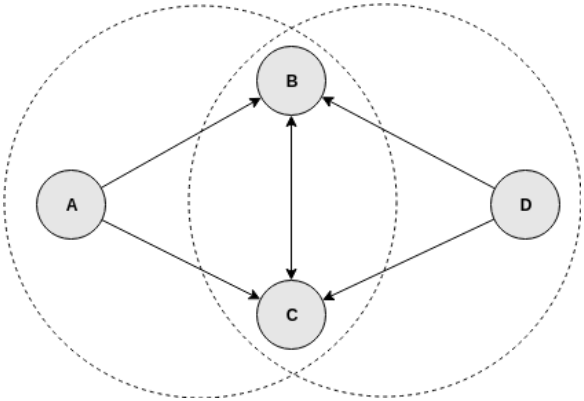

Figure 5.7 — A quorum intersection

5.4 Smart contracts

A smart contract is a piece of code that is executed by all parties involved and that changes the state of their accounts

A *smart contract*, as initially defined by Nick Szabo, is "a set of promises specified in a digital form, including protocols within which the parties perform on these promises" [27]. In other words, a smart contract is an ecosystem that presumes the following:

1. Set of conditions written in a programming language that can be checked and executed by a computer.
2. Program that explicitly executes the contract code regardless of the computer running it.
3. Distributed database of the asset ownership rights of the contract parties involved, that these parties consider the only valid source of data.

Taking a closer look at the smart contract, its lifecycle consists of the following stages:

- *Contact creation*
- *Audit and testing*
- *Accepting contract terms by the parties (signing)*
- *Adding the contract to a transaction*
- *Validation and execution by validators*
- *Comparison of the execution results and reaching consensus on updating the ledger*
- *Updating the ledger state (including the account states of the involved parties)*

Note that there is no need to store the full contract in the accounting system: it is enough to save its hash and provide its contents to validators for verification purposes.

When using a smart contract it requires the respective parties to accept its conditions as final. More general, a smart contract automates the distribution of digitized assets and its execution depends only on the conditions that have been

defined prior to its implementation. In its simplest form, it looks like a contract with strictly defined conditions which are signed by the involved parties. The most commonly used types of contracts include the following:

- *Timelocks (transaction confirmation is delayed for the specified time period)*
- *Hashlocks (assets can be unlocked if the hash function preimage is provided)*
- *Hash timelocks (assets can be unlocked if the hash function preimage is provided during the specified time period)*
- *Multisignature (performing operations of a particular transaction requires signatures by multiple predetermined parties)*
- *Atomic swap (an exchange of assets between two accounting systems; the peculiarity herein is that the exchange doesn't require participants to trust each other, and the transaction is either conducted inseparably or not conducted at all)*

The greatest advantage of smart contracts is that they allow setting conditions which are proven by mathematical and cryptographic principles.

Smart contract as a trigger

A smart contract can be described as a set of independent and non-contradictory conditions that change the ledger state depending on the identifiers, assets, and the data that is provided as input (Figure 5.8).

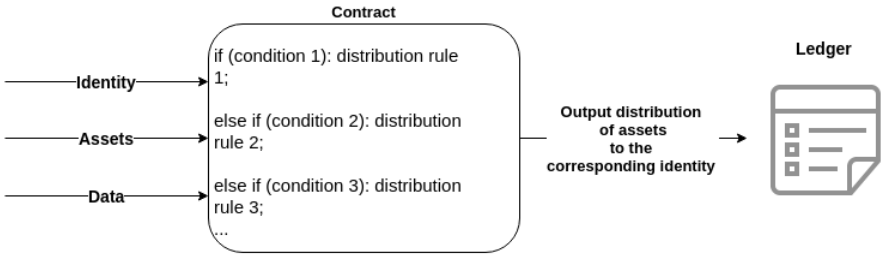

Figure 5.8 — Smart contract principles

Such a structure of smart contracts allows for the following advantages:

- *Completeness*
- *Ease of auditing and testing a contract before it is signed*
- *Ease of calculating fees*
- *Possibility to verify the conditions while keeping part of them confidential*

The first advantage lies in the fact that any such contract is guaranteed to be executed. Accordingly, the simplicity of the structure enables auditing any contract in an easy way and allows calculating fees prior to the execution.

Confidentiality

How is the confidentiality of smart contract conditions ensured? Any contract can be represented by the condition tree as in Figure 5.9. Only the root value is published in the corresponding transaction.

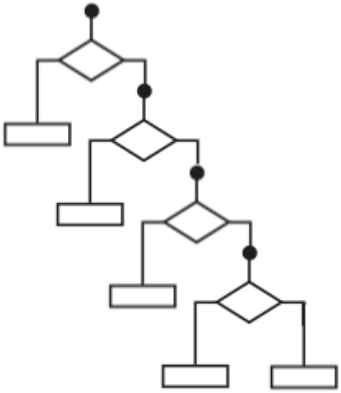

Figure 5.9 — Smart contract conditions in the form of a tree

Thereby, the necessity to prove that a contract is executed according to the specified conditions arises only once the contract was executed and the *Merkle branch* value proves that this value exists in the condition tree [28].

6 DATA MANAGEMENT

6.1 Main principles of data storage

Data is not stored on the blockchain-type structure for the purpose of scalability and meeting the GDPR requirements

In addition to managing and storing entities such as identity, transactions, and assets, it is also important to implement reliable data management mechanisms. The data may include open data that is provided by clients for performing specific operations on a platform (for example in the case of real estate trading, one must provide relevant documents, photographs, plans, permissions, etc.). For such data, integrity and availability are the two most important security services to provide for. The data integrity can be verified by adding its hash values to the corresponding transaction whereas data backups and storage on multiple servers can ensure the availability (Figure 6.1).

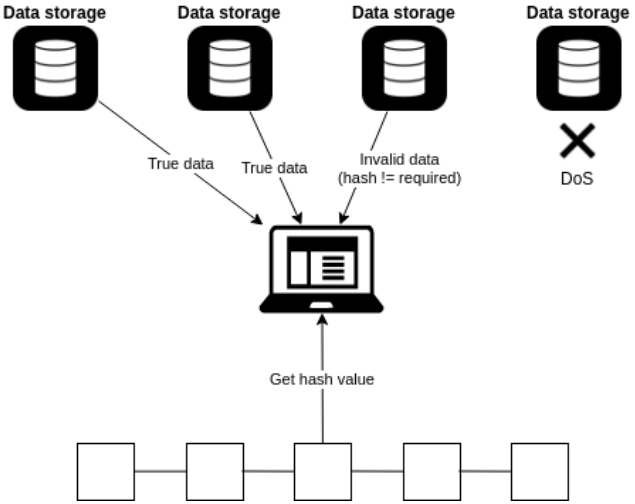

Figure 6.1 - Ensuring the data integrity and availability

The data also includes the users' personal data (personally identifiable information) for which ensure integrity and confidentiality is essential.

Personally identifiable information (PII) is any data that could potentially identify a specific individual. Any information that can be used to distinguish one person from another and can be used for de-anonymizing anonymous data can be considered PII (for more details, see section 11.1). Personal data must be encrypted with the keys of appropriate administrators (as the data must be available to its owners as well as to individuals who received appropriate permission).

The responsibility for processing and storing these details securely lies solely with the storage owner. In order to ensure the integrity of the data, an accounting system's transaction (e.g., to update the account details) may include the corresponding hash value. This ensures that personal data was processed correctly and has not been modified by an adversary (obtaining personal data from the hash value is a quite complex task). In addition, such data should be replicated and stored on multiple servers (principles of redundancy).

Another type of data is the private data of a client. If the system participant has decided to store his private keys on a remote server, then they should be encrypted on his side, and the encryption key should be available only to the client (it can be generated using a key and password; for more details, see section 8.2). To protect against attacks that use password dictionaries, a random salt value must be used to generate the encryption key.

All systems that store the listed data types must have their own sets of information security systems depending on the functions performed (and their security policies).

Data lifecycle

Data lifecycle includes the following stages:

- *Creation*
- *Access policy setting*
- *Storage*
- *Update*
- *Transfer*
- *Deletion*

Monitoring and control

Another important point in data storage is the data access audit. In terms of security, a reliable audit system is essential since it enables the monitoring of all the processes executed in a system, avoiding incidents, and responding to violations of security policies competently and promptly. Essentially, the data access control can be divided into the following three stages:

- *Placing the data into a system*
- *Configuring the data access policy*
- *Conducting audit*

In the first stage, the details of data processing and storing are already known (depending on the category this data belongs to).

In the second stage, the following details are specified: who has the right to access it, how these rights are proved, features of authorizing and authenticating clients, etc.

The third stage covers the time frame during which data is stored in the system. At this stage, all actions for receiving and processing data must be recorded, namely the following details: who and when requested access, which data was modified or deleted and for what reasons, all failed attempts to access the data as well as attempts to violate the security policy.

6.2 Data management

Data management includes receiving, validating, protecting, and processing the required data to ensure the main security services for its clients

In this section, we will describe the features of data management, namely how data should be processed and transferred.

Personal data management

Now let's analyze how the users' personal data is managed. Note that according to the GDPR (for more details, see 11.1), personal data cannot be transferred to other organizations without the user's prior permission [14].

Accordingly, the requesting party must first have the user's explicit consent and only then request the data from the organization storing it.

The question is, how is this permission granted and what is the information it should contain? First, the permission must be signed by the user whose personal data is being processed. By signing the permission, the user claims that he agrees with all the details of this permission (and it cannot be changed afterward since the digital signature guarantees the integrity and authenticity of the signed data).

Also, the permission must specify which part of the personal information can be transferred to a third party. In section 11.2, we describe the features of how one can gain access to only the permissioned subset of personal data while still being able to verify that this data has not been modified.

The recipient of this personal data is a third important component. As a security mechanism, a user can specify both the personal data and the recipient's name as well as her public key. Thus, the data sender can encrypt personal data with the public key specified in the permission, and no one but the owner of this key will be able to decrypt it.

Public data management

The main requirement for publicly available data is its integrity. This requirement can be met by adding the corresponding hash value as an identifier for this data. In this case, any client can request data by its identifier and verify the integrity of the received content.

Such principles are used by decentralized file sharing protocols (BitTorrent, IPFS [29; 30]) and can be applied to storing and managing data in financial systems, which, in turn, are components of Financial Internet.

7 DECENTRALIZED SYSTEM OF IDENTITY MANAGEMENT

Application servers utilize their user identifiers to identify them and provide certain services. To use a service, the user must first pass an authentication procedure, which may require her to provide proof that she is the one she claims to be. That proof can be any of the following:

- *Some knowledge (password, private key)*
- *Some material carrier (token, smart card)*
- *Some individual attributes (biometrics)*
- *Prove that she has been previously authenticated by a trusted third party*
- *Context (address, age, etc.)*

To prove the ownership of the rights in question, traditional methods usually rely on some user information and require the use of logins and passwords. A modern system, however, is focused on the usage of cryptography (similar to SSH [31]). Most often, the user's public key in such a system is used as an identifier. When accessing the application server services, a user pre-signs the request with her private key; in turn, the server verifies the signature value and provides access to the service (or rejects the request).

In each case, it is important that the identification system is fully reliable. We will further explore the basic principles of building a secure decentralized public key infrastructure, which will be the main source of information about users, systems, and application identifiers later on.

7.1 Architecture of a decentralized identity system

When designing a decentralized identity system, we solved a non-trivial issue: the integration of a decentralized technologies stack for the creation of a single source of identifiers for any subject and organization, whether it is an independent business or a global social network. Looking into the future, such a system should combine the following principles:

- *Multiple identity providers (regardless of their accreditation level)*
- *Real-time synchronization between identity providers*
- *Flexible level of trust of an application/user towards different identity providers*
- *Uniform (or even a single) identifier for multiple applications*
- *Combination of identity providers to increase recognition level (to extend permissions)*
- *Preservation of personal data confidentiality as well as its verification and transmission flexibility*
- *Real-time access to all events that happened to an identity*
- *Ability to store identifiers from any accounting system*

This section describes the general architecture of the decentralized public key infrastructure as well as the roles of components involved in building it.

Traditional approaches to building the public key infrastructure

Today there are two basic approaches to building a public key infrastructure (PKI): using a hierarchical model according to X.509 standard [32] or implementing a completely decentralized solution based on Web-of-trust [6].

The first approach is widely used in existing systems since it is quite effective in terms of performance: the process of obtaining, updating, and revoking a certificate does not require a lot of time. Moreover, such a model can quickly respond to any lower level certification authorities' keys compromise.

The standard X.509 describes a hierarchical model, where certificates of the certification authority (CA), end users, systems, and applications are signed by the superior CA. The only exception is the root certification authority, which is at the top of the hierarchy. The structure of the hierarchical model is shown in Figure 7.1.

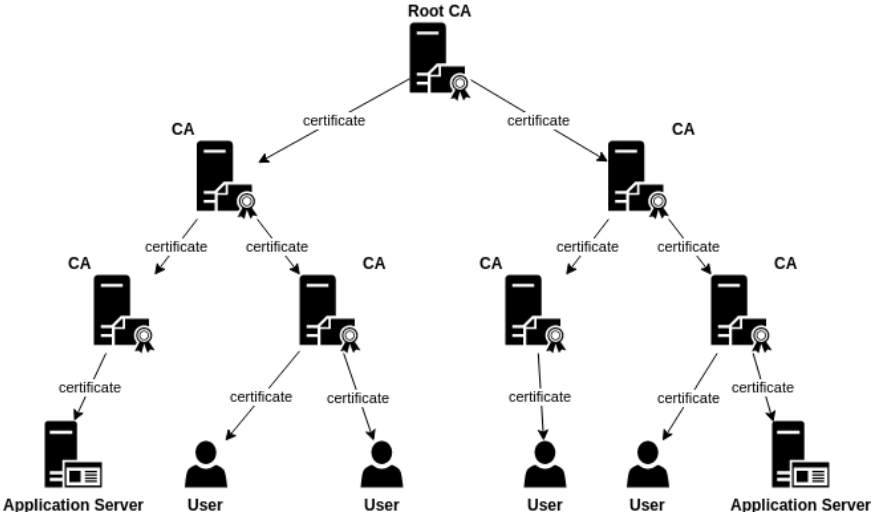

Figure 7.1 — Hierarchical public key infrastructure

As noted earlier, this approach has several advantages, but there are also some drawbacks:

- ❖ *Issue of obtaining certificates from different CAs for one entity (domain name)*
- ❖ *Issue of trusting certification authorities of all levels*
- ❖ *Necessity for a user to verify the entire chain of certificates (up to the root)*
- ❖ *Problems associated with synchronizing Online Certificate Status Protocol (OCSP) servers* [33]
- ❖ *Sole censorship which stems from the root CA*
- ❖ *Possibility of compromising the CA's keys (single point of failure)*

The main problem associated with the hierarchical model (the one which is first in the list above) is that clients need to go through the registration procedure in different centers that do not synchronize with each other. Thus, participants need to acquire a large number of certificates which are all tied to one identity. If the identification system is decentralized, the user will have a single accepted identifier confirmed by many independent providers. This also reduces the risk that identity providers collude and start creating fake identifiers.

The second approach, which was proposed for the OpenPGP protocol [], implies setting up a peer-to-peer trust network. The public key infrastructure based on web-of-trust is the most decentralized model as each system participant personally deals with certification and verification of certificates of all other system participants (Figure 7.2).

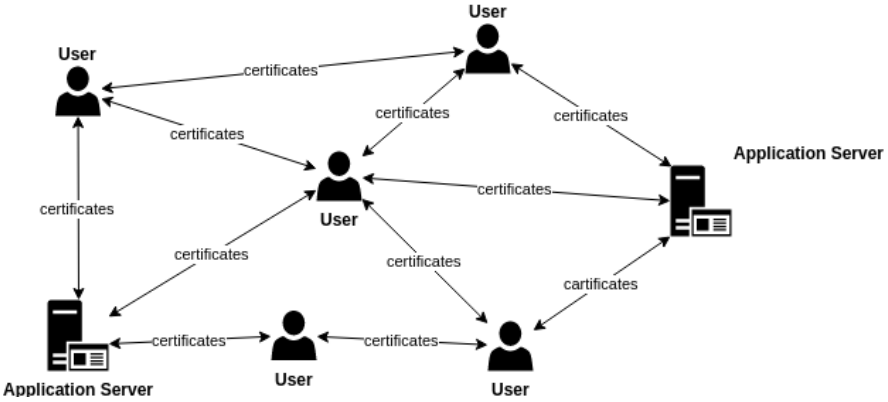

Figure 7.2 — Web-of-trust approach

This approach allows solving issues of the hierarchical model as well as improving the objectivity of information about user identifiers, but it is much less flexible since every action with a valid certificate (updating, revoking, etc.) should be carried out by all nodes storing it. Another limitation is the complexity of building a chain of certificates with a high final level of trust [34].

Decentralized PKI architecture

A decentralized identification and certification system consists of the following components:

- *Identity providers*
- *Registration modules*
- *Personal data storages*
- *Identity ledger*
- *Users and applications (systems)*

Schematically, the position of components and their relationship is represented in Figure 7.3.

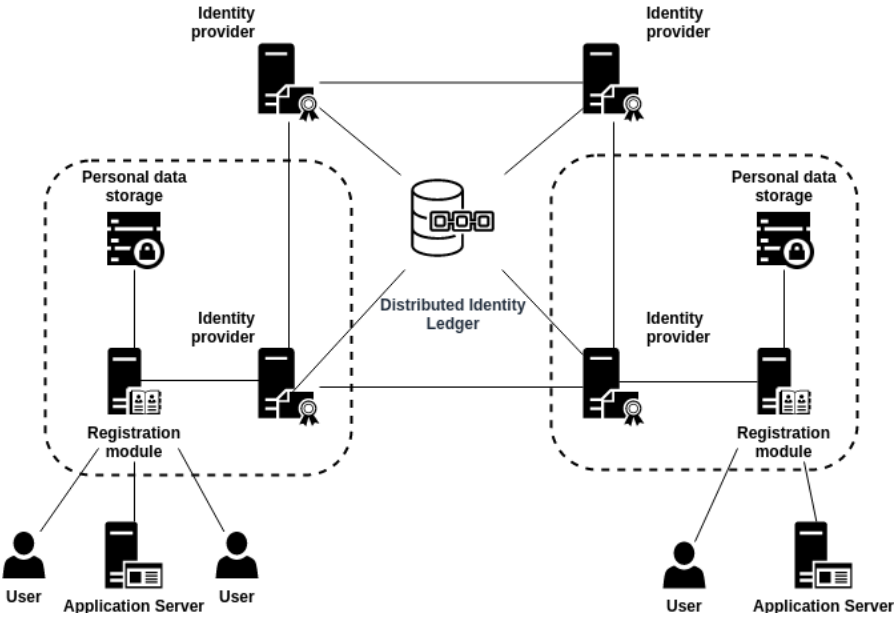

Figure 7.3 — Decentralized PKI components

Identity provider

An identity provider is one of the main components in the public key infrastructure. Identity providers are engaged in issuing identifiers for validating rights of users, systems, and applications. Their role is to maintain an immutable ledger of events with their timestamps.

To create an account, the identity provider receives the necessary data (public data and public key) from the registration module and then forms transaction that confirms the identifier and signs it with a private key, which confirms its authenticity (Figure 7.4).

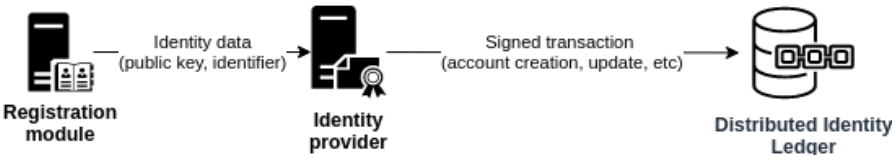

Figure 7.4 — Identification procedure

The identity provider's public key is available to any system participant. The client's account is placed in the public distributed ledger.

> *NOTE*
>
> *Public ledger means that all the data on the ledger is available for all (i.e., neither party should have any permissions to gain access to the ledger).*

Registration module

The registration module is the component that is responsible for registering users and accepting identity requests. The registration module specializes in identifying end-users, systems, and applications, and it can both be external as well as associated with a specific identity provider. In this case, the level of trust towards a particular registration module is determined by each identity provider individually (e.g., registration module can conduct different types of identification for medical and bank organizations which are separate identity providers). The process of integrating an external registration module (also known as the KYC provider) with an identity system is described further in the book (section 10.4).

The registration module conducts the initial identification and registration of users. To obtain an account, a user must request the registration module directly. The registration module handles the user's request and the data provided in it. Once processed, the registration module sends the request to the identity provider or if the information provided is not sufficient, additional data is requested from the user.

The important point here is that among the data that is processed by the registration module, there is also the user's personal data. Since personal data

cannot be transmitted to third parties (and processed by them) without the user's explicit permission, it is necessary to ensure confidentiality all along. How confidentiality is ensured along the way is also explained in more detail in section 11.2.

Since the registration module (and its owner) is fully responsible for the storage and processing of the user's personal data, a CISS (complex information security system) should be created and a corresponding security policy containing the principles of storage, processing, and backup of information should be established.

Personal data storage

The personal data storage is designed to store data that is provided by users to the registration module. The user data is encrypted with registration module's key, and the corresponding registration module has direct access to it. If the user's personal data is updated, then the registration module also needs to update the record in the personal data storage (and send the updated hash value to the identity provider).

Third parties can contact the registration module to obtain the user data (for example, to verify whether this data matches the hash value of that in the account). In this case, an applicant is required to obtain permission from a user to receive and process her personal data. The registration module checks the request, and if the user's permission is received, the registration module retrieves the data from the storage and sends it to the party that requested it via a secure channel.

Distributed identity ledger

The identity ledger is a distributed ledger which is stored on each identity provider. Actions with identifiers are organized in an orchestrated, linked list of transactions, where each next set of data refers to the previous one. This ensures the integrity of the identities that are added to the linked list and renders backdating identities impossible. Every action that occurs with the user's identity (issuance, updating, revocation) must be initiated by a transaction signed by one or few of the identity providers.

Users, systems, and applications

Public key infrastructure is used by users, systems, and applications to ensure the security of interaction between them. PKI is needed to provide the following security services to users, systems, and applications:

- *Confidentiality*
- *Integrity*
- *Authenticity*

Confidentiality is ensured through asymmetric encryption (simply put, this is achieved through a shared secret) using public keys of the system participants. Thus, any two system participants can organize a secure channel between them and exchange data without the threat of disclosing data to third parties. PKI eliminates the man-in-the-middle (MITM) attack in such an interaction [35].

The control of integrity and authenticity of the transmitted requests and messages is provided through using digital signatures. System and application servers provide corresponding services to users based on users' identifiers. Thus, when accessing the server, a user signs the transmitted request. An application server then verifies that the signature is correct and that the user is who she claims to be. For this, a public key certificate is used.

7.2 Initial identification process

Identifiers are used to associate specific users, systems, or applications with specific public keys. Identities are used to verify the authenticity of users and applications as well as to control access (authorization). As mentioned in section 7.1, identifiers are confirmed by identity providers.

Account structure

The account structure consists of two main aspects: information about the identity providers (who confirmed the identifier) and the identification object.

Account contains the names of the identity providers that confirmed the user identifier (this could be the name of an organization to which a particular identity provider belongs). And account also contains the public keys of the

identity issuer. Another parameter is the level of user identification, namely the amount of data that was verified by the identity provider with respect to a specific account (e.g., email, email + phone, etc.).

Account also includes the identifier of the subject (hash unique email or phone number) to whom the certificate was issued. Also, it contains the hash value of the user-provided personal data (the process of providing data is discussed below). When adding identity to the identity ledger, the registration module will not be able to modify the stored personal data of a particular user on its sole discretion since any modification can be detected and proven quickly. This value can also be used by third parties to verify that the registration module has authenticated the user correctly.

Extension field contains information about the scope of keys (i.e., what the keys will be used for) linked with the identifier.

Noteworthy, all events associated with an account should be added to the shared history in the form of a transaction. Examples of such account-related events are the verification procedures by identity providers, updating and deleting account data, etc.

Account creation process

To create an account, an identification subject requests registration module for the initial identification and registration. The registration module asks the subject to provide all of the required personal data. The data is then processed (identity is being verified, namely that certain public key is associated with email, phone, name, etc.) and stored in secure storage (the storage of data can be performed by a separate entity as well). In addition to storing the data, the entity also provides two services: gives assess to this data to the authenticated user and also, if needed, shares the data with the third parties if the user has given permission to share it.

Once registration process is completed, the registration module provides the identity provider with a hash of the received data, the subject's identifier (or its hash value), and the public key (or set of public key). The identity provider receives the data and initiates the account creation. The process of issuing the identity is shown in Figure 7.5.

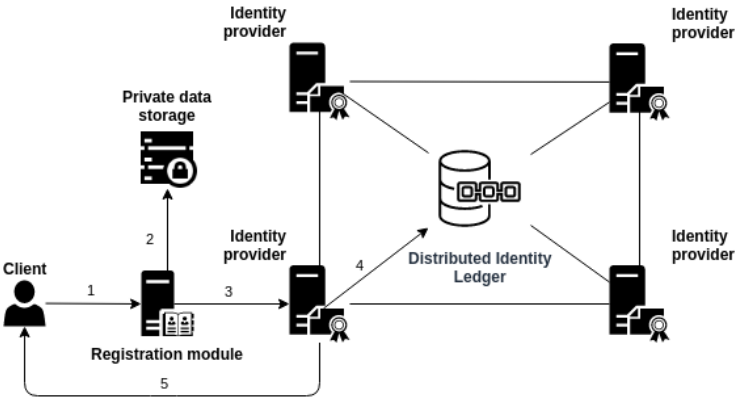

Figure 7.5 — Account creation

The process of issuing an identifier consists of the following steps:

1. Client sends her personal data and a generated public key to the registration module. All the data is transmitted encrypted (asymmetric encryption).
   1.1. Registration module processes the request (carries out the user identification; for more details, see section 7.1). Since the goal of the key issuance is specified in the request (in *extension*), different data sets may be required for identification.
   1.2. If the data provided by the user is insufficient for the registration module, it can request additional data. If the user provides the requested data, the registration module proceeds to step 2; otherwise, the user's request is rejected. Also, the registration module checks that the client owns his identifier (sending an email confirmation letter or a calling a phone).
2. Registration module encrypts the user's personal data and puts it into a secure container. Only the registration module has direct access to this personal data, and if a third party wants to access it, then it must get the user's permission first.
3. Registration module transmits the following data to the identity provider: the user's identifier (for example hash of email), the user's public key, and the hash of his personal data. Note that all the data is

also transmitted in an encrypted form and signed by the registration module (thus eliminating the risk of data-tampering by any third party). Identity provider processes the data received from the registration module, creates an account, and signs it with its private key.

4. Identity provider creates a transaction that contains the account creation operation and propagates it amongst the nodes (CAs). The nodes verify the transaction (that signature is correct and data is well formed.) and reach consensus about updating the distributed identity ledger.
5. As soon as the transaction is added to the identity ledger, the identity provider returns the confirmation to the client. Now, the user can use his key pair to interact with other users, systems, and applications.

Figure 7.6 below shows the interaction flow of the components of the identification and certification system.

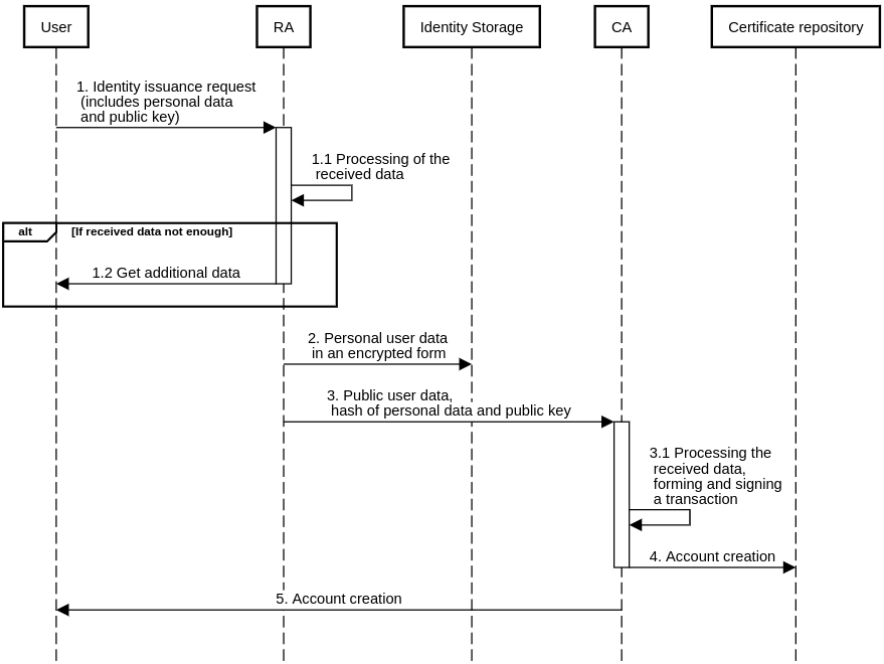

Figure 7.6 — Account creation flow

7.3 Identity ledger and identity verification process

Correspondence of each identifier is verified based on all operations associated with each particular account

In this section, we describe the principles of building the identity ledger construction and its identity verification procedure.

Principles of building the identity ledger

As mentioned earlier, identity storage is a distributed ledger which is required to achieve consensus between the validators of the system (the particular formation principles may differ depending on the number of validators). To obtain identifier information, platform clients send corresponding requests to the identity platform and receive up-to-date information on the user's identity and status.

During the system operation, transactions of participants associated with specific accounts are added to the ledger (creation, updating). The purpose of the ledger is to reliably timestamp the account-related events in real time. In fact, each operation is only verified for the compliance of its structure with the protocol rules and that the signature in it is correct. This means that no additional checks are carried out, and it is only important to compare the events (voices related to account changes) and to know who initiated this event (Figure 7.7).

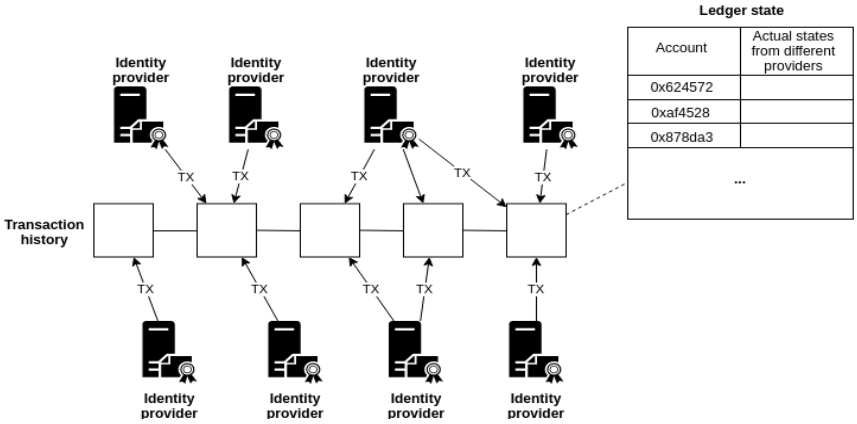

Figure 7.7 — Identity ledger forming

Identity verification

Every participant of the decentralized identification system can store a complete history of transactions locally, thus increasing the complexity of its unauthorized modification. At the same time, everyone can receive complete information on a specific identifier in the system: by whom and when this identifier was issued, by whom it was further confirmed, how the subject was identified (remotely or in person).

Recall that every account contains information about the identifier confirmation by various identity providers. It is also specified how the initial identification and the reidentification of subjects was carried out. Depending on this information, users, systems, and applications decide by themselves how much to trust a specific identifier. If a network participant has his identifier confirmed by a vast majority of independent identity providers who claim that he showed up personally, showed the required documents, etc., then the chances of this particular identifier belonging to a particular network participant are much higher.

On the contrary, if a new participant has just appeared on the network, has been identified only by one identity provider and has provided only a small set of initial data, then the level of trust to his identifier won't be as high.

The obvious result here is that the state of information about the correspondence of a particular key to a particular identifier will not be the same for the entire network—this would rather not be some constant value; instead, it would be a sequence of updates that are made by different registration modules. Some identity providers may, theoretically, provide incorrect data about client keys. However, if the majority of providers are honest, most of the information will be correct. This can be used by accounting systems and individual applications. In this way to verify identity, the verifier receives only a sequence of updates about the identifier from the system and then based on the trust levels to involved registration modules, the verifier makes a decision about the identity status.

There is another threat which is related to the fact that users can provide different data to different identity providers, resulting in some confusion on the network. However, in such a case, users are putting spikes in their wheels, and the result would be that they will find it hard to receive services from the accounting systems.

7.4 Updating identifiers, public keys, and credentials

Updating personal data and keys is essential in any identity system. The swift and correct execution of these processes determines the platform's overall efficiency. In this section, we will describe the following processes: updating personal data, updating account identifies and updating the subject contact information.

Account renewal

To renew an account, a user must appeal to the registration module with a corresponding request. Since the user's personal data is already stored in the registration module (it was placed to (and consequently stored in) the personal data storage at the initial identification stage, while its hash is stored in the certificate), the user does not need to carry out the identification procedure again but only sends the request details to the identity provider.

Figure 7.8 depicts the interaction flow between the PKI components during the identifier renewal process.

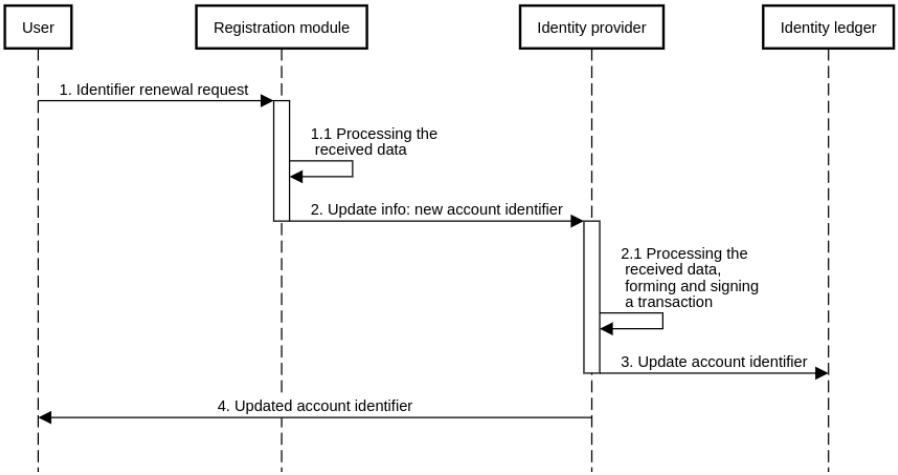

Figure 7.8 — Account renewal flow

The interaction flow between the PKI components is as follows:

1. User sends a request to update the account identifier to the registration module. The request contains the user identifier and the update value.

1.1. Registration module processes the received data. Note that the registration module does not need to reidentify the user since all personal data is already in the storage.
2. Registration module sends a request to the identity provider to renew the specific account identifier.
2.1. Identity provider processes the request and, based on the received data, generates the corresponding transaction and signs it.
3. Identity provider sends the transaction containing the updated account identifier. Upon confirmation, this transaction updates the corresponding account in the identity ledger (applicable to all identity providers).
4. Identity provider informs a user that the respective account has been successfully updated and returns it to her.

Account re-key process

An account re-key process implies that a user changes the public key (or adds new signers) for which the certificate already exists. A user generates a new key pair and applies to the registration module (which already has the user's personal data). The registration module sends the account update request and the new public key value to the identity provider. Initially, the identity provider must invalidate the previous key. Then, the identity provider creates and signs a new transaction that contains two operations: revocation of the old public key and addition of the new one. After the received transaction is confirmed, the rest of the identity providers update their ledger states correspondingly.

Personal data update process

The personal data update process is related to changing the account subject's personal data. In this case, an identification subject refers to the registration module with an updated set of its data in the request. The registration module performs the identification based on the received data, deletes the obsolete data from the personal data storage, and adds the updated one. After that, it computes the hash of the new dataset and sends it to the identity provider along with the account update request. The identity provider

processes the request, updates the account (which is identical to the old one except for the hash of the subject's personal data), signs transaction, and sends it to other providers. Figure 7.9 shows the diagram of the interaction flow between the PKI components during the subject's personal data update process.

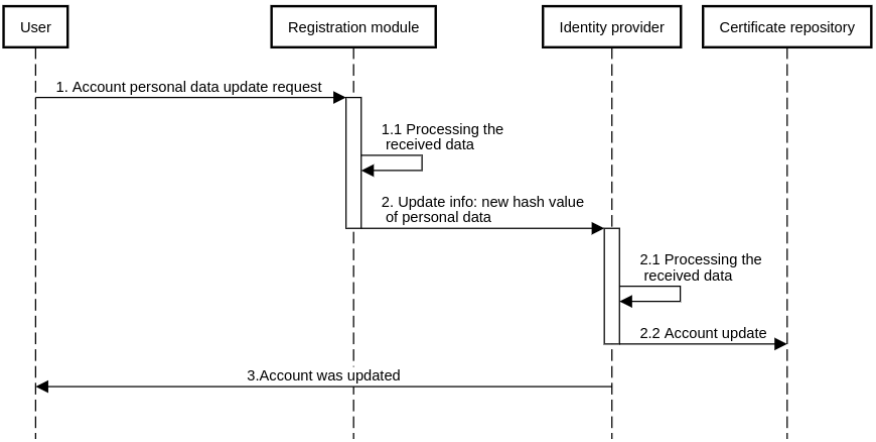

Figure 7.9 — Personal data update flow

The interaction flow between the PKI components is as follows:

1. User sends an account update request tothe registration module (along with the new set of the subject's personal data).
   1.1. Registration module processes the user's request. In this case, registration module must perform another identification, namely, to verify that the provided attributes are valid for the actual account.
2. After this reidentification is carried out successfully, the registration module sends a set of the specific certificated subject's personal data to the personal data storage. After the new dataset is being put to the storage, the registration module computes its hash.
3. Registration module sends the request to update the specific account to the identity provider, which includes the subject's current identifier and the updated hash of the subject's personal data.

3.1. Identity provider processes the request and based on the data received, it creates and signs the transaction.

4. Identity provider sends a transaction containing the update operation. Once the transaction is confirmed, the corresponding account will be updated in the identity ledger (for all identity providers).
5. Identity provider informs the user that his account has been successfully updated and returns the updated certificate.

For all the described processes, it is important to note that all the account change operations (identifier renewal, keys and personal data renewal) should be initiated by the corresponding transactions, the history of which is stored at each validator node. As a result, each of the platform participants sees what actions were taken on the specific account and by whom they were carried out, which allows for an objective assessment of validators' actions and getting the current status of identifiers.

7.5 Features of a decentralized PKI

Using the approach described in the prior section allows the setup of a shared infrastructure between users and particular application servers. The key feature of such a structure is the use of a single identifier to receive services from all application servers in a decentralized system.

In this section, we will consider first how to apply to various application servers using a single identifier. Secondly, we will describe how to provide additional identification if the data provided by a user is not sufficient for an application server or if this server wants to make sure that the identification procedure has been carried out correctly.

In addition, we will consider the basic security aspects of such a system and the features of its operation in case the keys of end users, applications or the top-level components have been compromised.

Identifier lifecycle

Let's take look at the possible stages in the lifecycle of an identifier and of the account to which it belongs (Figure 7.10).

- *Initial creation on a client device*
- *Initial verification by one of the providers*
- *Update of parameters (e.g., signers, weights, etc.)*
- *Increasing the identification level*
- *Verification by other identity providers*
- *Revocation of a key*
- *Update (identifier, key, personal data set, etc)*

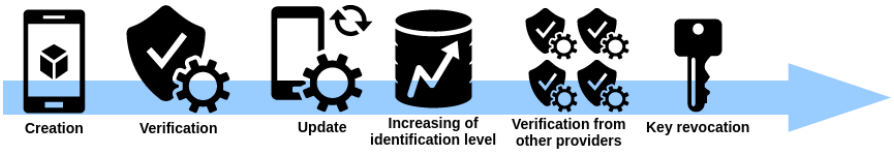

Figure 7.10 — Identifier lifecycle

Using a single identifier to access separate application servers' services

As noted earlier, each user is initially a client of one of the identity providers. Taking this into account, all identity providers share a network and a distributed database built as an ordered set of transactions. To access a specific application, a user must first pass the authentication procedure. To do this, he signs the request with his private key and sends it to an application server. The server verifies the signature value and the user's public key certificate.

The certificate verification procedure is described in section 7.3; however, the verification of the user's certificate by another user may differ from the certificate's verification by particular applications (for example, in banking or medical applications).

Typically a user does not provide specific data, such as payment details or medical insurance data, when he passes the initial identification procedure and creates of account. Moreover, a particular service may not trust the identification procedure performed by some other organization.

To confirm the user identifier (as well as to extend the user rights and permissions), any application service or registration module can conduct additional user identification to confirm or add the identification data.

The data a client provides varies depending on the level of permissions she wants to receive. For example, to confirm the ownership of an email, she needs to provide only an email address; for the registration in a social network, an email address, a phone number, etc. are required; to open a bank account, she would need to provide all of the above + her passport details and maybe even meet a banker in person (Figure 7.11).

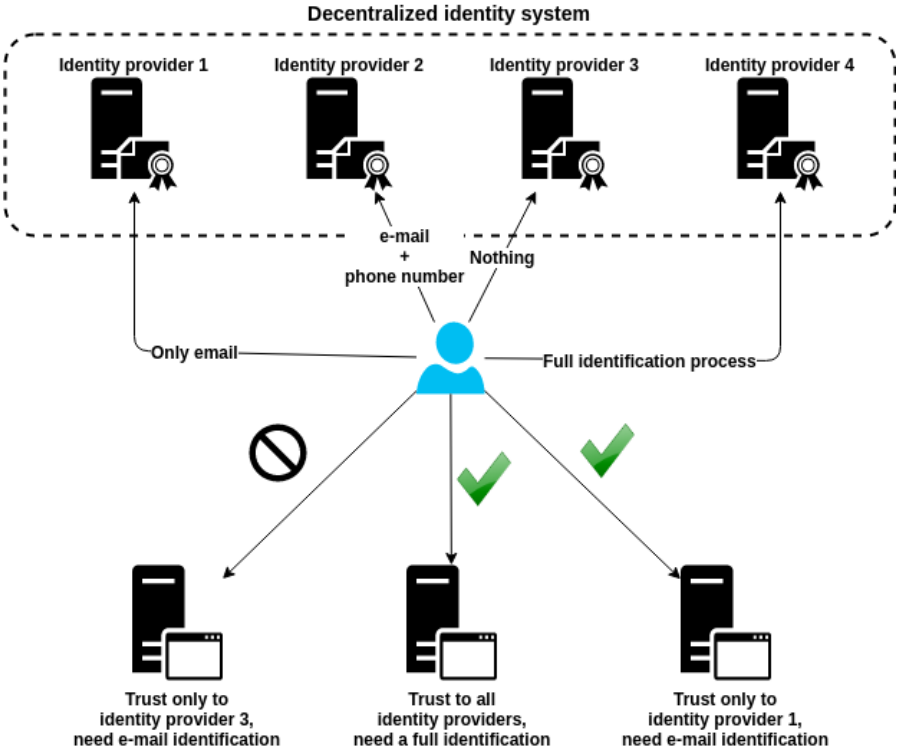

Figure 7.11 — Access to different services depending on the level of identification

A user can increase his identification levels to obtain additional permissions and to extend the network of applications that can provide their services to him.

Also, the user can extend his account settings: add signers and configure their permissions and weights. Note that every accounting system or a separate application independently determines the public keys of the signers it trusts. In such a case, a decentralized identification system acts as an immutable source of information about all actions associated with the user account (and its corresponding signers).

Compromise of the end user's private key

In the case of the user's private key compromise, he applies to his registration module with the certificate revocation request. At this moment, a reliable user's authentication is very important in order to prevent an attacker from revoking the client keys.

After the user has sent the certificate revocation request to his registration module and confirmed his identity, the identity provider forms a transaction with the key revocation operation and transfers it to the other identity provider.

Note that in such a case, the certificate revocation operation hinges on the trust towards the identity provider that confirms that 1) this key revocation operation is not an attack and 2) that a user has been reliably authenticated and actually wants to revoke his keys.

Compromise of identity provider's keys is no different from the user keys compromise: the provider informs that its keys have been compromised, and each system participant defines independently how much trusted this information is and deals with the transactions signed by these keys with utmost caution.

7.6 Scope of application

Digital identity will be used virtually everywhere

Decentralized PKI for security services

The usage of cryptographic mechanisms enables the confidentiality, integrity, and authenticity of any data transmitted. However, to securely execute

these operations, you need a mechanism for exchanging public keys between participants and for confirming the key ownership of a specific person.

The key question is, how can one participant make sure that the public key he receives does belong to another specific participant? What prevents a man in the middle from intercepting the transmitted public key and replacing it with his own?

For this, the public key infrastructure (PKI) is used, which allows linking the participant's data with the public key and provides a mechanism for the quick verification of this binding authenticity. The primary object in a PKI is an identity and linked public key, which is used for the following:

- ❖ *Managing the flow of digital documents*
- ❖ *Verification of the authenticity of web applications*
- ❖ *Signing files and software elements*
- ❖ *Messaging*
- ❖ *Running online shops and financial applications*
- ❖ *Passwordless authentication*
- ❖ *Cryptographic proofs of asset ownership*
- ❖ *Cryptographic proofs of data provenance*

Secure email and document flow mechanisms use a digital signature to ensure the integrity of transmitted messages and documents (protocols such as MIME [36]).

Public keys are also used for messaging between a client and an application server. In this case, the custom software contains the public keys of the visited resources and verifies the signed responses. Accounting systems must require their users to sign requests for every message. This provides for a higher level of security compared to a sessions mechanism (for more details, see section 9.1).

Digital signatures and public keys are also used to sign executable files (or scripts). This guarantees the immutability and copyrights of the software.

Currently, an increasing number of messengers use asymmetric encryption between clients. This applies not only to the decentralized solutions such as BitMessage or Tox, but also to Telegram, WhatsApp, and Viber, which all

support end-to-end encryption. It also requires a reliable system that binds identifiers with public keys.

In many countries, a digital signature is already legally binding and thus as credible as a handwritten one. Therefore, many financial institutions already support the certification of certain actions on behalf of their clients. Moreover, many banks are the identity providers themselves.

Single sign-on

Single sign-on (SSO) is a mechanism that implies the one-time authentication of a user with further access to the necessary resources [37]. In its pure form, there is an application that stores user's keys/passwords which after the user has authorized, automatically fills in password fields with this data or signs messages and authenticates the user in different services.

The concept of decentralized PKI extends this approach. The user's application can be the repository for storing her keys, while the decentralized structure is the source of information about them. In such a scheme, every user owns a particular uniform identifier and a particular key (set of keys). To access the accounting system services, she sends her identifier and signs the corresponding request. The service verifies that the identifier has the appropriate permissions and that the signature is obtained using the appropriate keys.

From the user point of view, she launches the application, authenticates, and accesses her keys. All further requests to accounting systems are signed by the application that stores the keys (Figure 7.12).

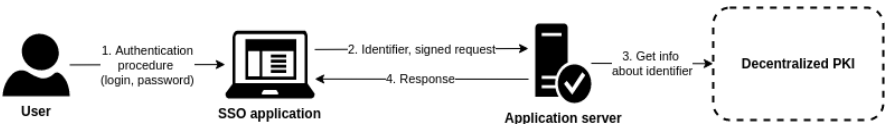

Figure 7.12 — Authorization process

Digital identity for Internet of Things

To support the normal operation of a system, IoT also requires the use of robust identification mechanisms [38]. Each individual sensor must properly

interact with the rest of the system. For secure communication between system components, they must exchange encrypted messages while being able to verify the data integrity and authenticate the sender. This is only possible by having a reliable public key infrastructure.

As the number of individual components grows, it becomes increasingly difficult to establish a reliable mechanism for identifying data and data sources. In this case, there is a number of tasks that can be accomplished with existing and already functional systems:

- *Managing the security policy as well as component roles*
- *Maintaining the rules for an automatic granting (and revoking) of the access rights to the component*
- *Automatic permission verification to ensure the access is following the system's security policy*

Figure 7.13 shows how IoT components are configured and interact based on the decentralized identification system.

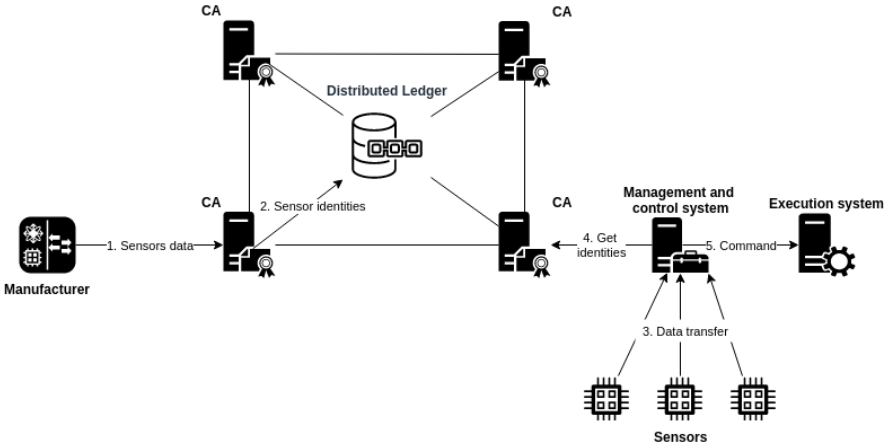

Figure 7.13 — Digital identity for Internet of Things

The interaction of IoT components on top of the decentralized identity system includes the following steps:

1. Sensor manufacturers provide information about devices. Instead of

the personal data hash, they provide the hash of the documentation that describes the sensor. In this way, each sensor must be a separate identifier. The sensor is delivered with the pre-generated key pair that in some cases can be regenerated manually.

2. The registration module receives information about the device and its public key and creates a new account. After the transaction (including the operation for creating the account) is confirmed, the sensors can be used, while the authenticity and integrity of the data received from them can be verified.
3. The control system receives information from the sensors and sends the corresponding commands to an execution system (one example of such a system is the car's on-board computer).
4. The control system submits a request to the registration module to retrieve the sensors' identifiers.
5. Based on the information received from the identity ledger, the control system can make an assumption on whether the received data is authentic and complete.

8 KEY MANAGEMENT

8.1 Approaches to key management

Users should be free to choose the way how their keys are stored, fully understanding the benefits and drawbacks of each approach

Key management

Private keys are confidential data which, it is presumed, can be accessed (and, correspondingly, processed) only by its owner. Let's consider the key lifecycle in order to analyze the stages at which its confidentiality must be ensured and how keys are safeguarded along their lifecycle.

The key lifecycle stages are as follows:

- *Generation*
- *Waiting (storing)*
- *Active state (formation of keys used for encryption and signature)*
- *Post-active state (decryption and signature validation)*
- *Destroy*

In a perfect world, the key owner is solely responsible for all the processes: from processing and creation (generation) to redemption. Noteworthy, the reliability of the access to the accounting system services depends directly on the security of users' keys. Thus, if a user chooses a weak algorithm or unreliable method of generating random sequences, keeps her keys in a plain form, or uses vulnerable software, she has to accept the resulting risks.

An increasing number of systems that strongly rely on cryptographic keys usage introduces a new problem becoming more and more critical: storing these keys securely. That being said, keys are currently being used not only for reliable document management mechanisms but in everyday life as well, for example, to conduct electronic payments—in the near future, the vast majority of objects and processes will be managed digitally via cryptographic keys.

Let's consider the basic approaches to key management.

- *Keys are stored and processed on the user's device*
- *Keys are stored and processed on a remote server*
- *Keys are stored on a remote server but processed on the user's device*
- *Usage of multisignature and the combination of previous approaches*

Keys are stored and processed on the user's device

The simplest approach to key management implies that the user's keys are stored and processed directly on their device. An advantage of this approach is that a user does not need to trust the processes of her keys storage and processing to the third parties; instead, the user is solely responsible for these processes. In other words, this method is considered safe in the context of avoiding dependency on a third party and the related risks. However, it is also associated with several disadvantages.

The most significant disadvantage is that the device on which the user's keys are stored can get lost or broken. In this case, the user will unequivocally lose his access to the accounting system's services unless the keys have been backed up before. In the case of the user's device theft, the keys will be compromised (an adversary will be able to access them and use for their own purposes).

Another disadvantage is the difficulty of transferring keys to another user device. To do this, a user must either transfer them over the network (which is not recommended) or manually enter them on each new device (the need to transfer keys manually significantly decreases the convenience of key management in the accounting system).

Keys are stored and processed on a remote server

The case when keys are stored and processed on a dedicated server is one of the simplest: the client software does nothing except that it sends requests to the server, which, in turn, processes the requests with the user's keys, for example, when signing a transaction.

One of the advantages of this approach lies in the fact that users don't need to store keys on their device; correspondingly, its loss or damage won't result in losing access to the keys. Even if a user forgets the password to access the

service, he will still be able to use the accounting system services (it is similar to how traditional systems work: you may lose your phone or credit card, but you do not lose funds, and you restore your account).

It is noteworthy, however, that mechanisms for restoring access to the accounting system services are possible to implement in all the listed approaches to keys storage and processing. Yet this task is much easier to implement when keys are both stored and processed on a remote server.

Of course, this approach has its drawbacks. The first and the most significant one is that users do not have control over their keys (a service can potentially (and technically) dispose of them at its choice: sell them to a third party for example. Also, users will lose access to their keys in case of service denial. Moreover, if the servers controlling the service do not use robust enough security measures—such as encryption or storing different pieces in different places (i.e., secret sharing)—then an attack on the server could also lead to the loss of access to the accounting system service. Therefore, if you decide to choose such a key management approach, then it is recommended to take the relevant risks into account and to make a backup copy of your keys in advance—this may help in case the service fails to respond; however it won't help in case your keys are compromised by hackers who gained access to the server.

Keys are stored on a remote server but processed on the user's device

The third approach implies that keys are stored on the server but in an encrypted container, and only the user has access to this container since it was created and encrypted on his device. In this case, the user first encrypts her keys with an encryption key generated from a secret that is known only to him (usually a password) and then sends this encrypted container (i.e., encrypted user keys) to a remote server. Hence the server doesn't have access to the user keys and cannot process them; it just stores them.

The client-server interaction is organized as follows: a client submits a request to the server to receive an encrypted container; the server returns the container, and then the client decrypts it by using the corresponding key.

Advantages of this approach are obvious. Unlike the previous one, the service does not process users' keys and users can access their keys from any

device at any time. Simultaneously, the loss of the user's device won't result in the loss of the keys since they are stored on a remote server (unless the secret which is used for encryption/decryption (e.g. user password) is lost). Note that potential attacks on the server don't make much sense for hackers since they won't obtain the user keys but only an encrypted container which only users know how to decrypt (i.e. keys won't be compromised). The only thing an adversary can potentially do is cause a denial of service, which means that users are unable to gain access to their keys; however, this issue can be solved by increasing the number of servers on which users store an encrypted container with their keys.

Note that in the previous approach (where the keys are fully managed by a server), the risk of user keys being compromised increases as the number of servers that store and process these keys increases. However, when keys are stored on a server but processed on the user's device, the impact of increasing the number of servers is positive since it decreases the probability of a user losing access to their keys resulting from a service denial.

Noteworthy, if the user's keys are compromised when using this approach, then the blame is on the user (it means that the compromise is at the stage of key processing, or the user has disclosed the secret).

In conclusion, this approach is convenient and safe assuming that a user processes her keys and chooses her encryption secret wisely.

Usage of multisignature and the combination of previous approaches

Another option for a considerably more secure way to access the accounting system services is to combine the above-mentioned approaches by using a multisignature mechanism instead of managing assets via a single digital signature key.

What so distinct about this approach? Here, access to the accounting system services is tied to several key pairs (only n of m keys are necessary to receive the service). The storage of these keys can be distributed (for example, if the 2-of-3 multisignature is used, then the first key can be stored on a phone, the second one on a PC, and the third one on some centralized third-party storage. In this case, the loss or compromise of one of the private keys would not result in the user no longer being able to use the service. An adversary wouldn't have any

use of controlling a single private key. If one of the three private keys is lost, the user still has access to the other two, which is enough to access the accounting system services.

This approach is the most secure in the context of preserving access to the system's services. Still, there are also disadvantages. One of the drawbacks is that transactions dealing with an address that is controlled via several keys are much bigger in size than regular transactions (in most accounting systems, multisignature is for now a list of single signatures rather than a single value obtained by using several private keys).

Another limitation is that not every accounting system protocol supports the multisignature mechanism. Therefore, this approach is used only in some systems.

Summary

Keys are stored and processed on the user's device	Frees a user from trusting a third party in the context of the key management, but he is solely responsible for his keys at all stages of the keys' lifecycle and the user fully carries all risks associated with the loss or damage of their personal device.
Keys are stored and processed on a remote server	Fully takes the responsibility of managing the keys from a user while forcing the user to take risks associated with trusting a centralized third party. Note that the user is still responsible for managing her own password for accessing the service.
Keys are stored on a remote server but processed on the user's device	Removes the responsibility from a user to store keys but does not remove the responsibility to process them. This approach allows a user to access keys at any time while having to face the risks related to a server failure. In this case, however, the user password is a cornerstone of the system and losing it may entail the loss of access to the keys.

Usage of multisig-nature and the combination of previous approaches	It is the most secure approach while not the most convenient for the end user. It may be limited by the architectural features of the accounting system used.

8.2 Key server

The correct approach to the key server design is that it doesn't have access to the user keys: they are encrypted on the user's device and never transmitted over the network.

One of the options for the key storage is to store them on a remote server. At the same time, it is important to build a key storage model in which neither the server owner nor an attacker can gain access to user keys.

One of the security schemes for such an approach is to put the key in a protected container and provide access to the container only to an authorized party. The first task is solved by encrypting the private keys, the second by building a reliable model of authentication of the party by the server.

This section describes the scheme of key storage and authentication of the party by the server. The described scheme avoids the processing of secret data by the key storage server, avoids the use of the session mechanism (which is susceptible to intercepting the session identifier and gaining access to the session) as well as uses an authentication mechanism that does not require a user to provide data that could compromise his keys or passwords.

Issues with traditional approaches

The more business processes occur on the internet, the more important the security issue becomes. Predominantly, traditional scheme of work presumes that internet services are responsible for the storage of the user sensitive data (such as passwords), resulting in threats at all stages (transfer, processing, storage).

- *Password can be compromised if the communication channel isn't protected*
- *Phishing attack risks (the channel is secure, but a user is connected to a fake server)*
- *Passwords can be compromised because they are processed in an explicit open form*
- *Administrators can sell user secret data to third parties*
- *All user passwords are very often stored on a single server, hacking which results in the compromise of all user passwords at once*

Note, that since passwords of all users (or password hash values) are all accumulated "in one place", exposing it is much more attractive for a hacker (even if an entity maintaining the server uses the most robust and effective security measures) than the individual user device which is not that much protected.

The traditional approach involves the union of an application server with an authentication server (application server stores and processes an authentication data). In general, an application server stores user password hash values, and the process of user authentication is as follows. A user provides their password and after the successful authentication, a session with this user opens and she receives a session identifier. Subsequently, a user will stay within a session to receive the requested data by attaching this identifier to each message to the server. Such an interaction is convenient and proves to be effective in a number of cases, but it is prone to various attacks which may occur at different stages (Figure 8.1).

The first vulnerability lies in that a user must pass the value of his password for authentication. If the channel between the user and server is not protected (HTTP is used instead of HTTPS), then an attacker can easily see the data transferred and intercept the password.

After receiving the password, the server processes it in an open form (calculates its hash value). Therefore at this stage, a password may, theoretically, leak (whoever is having access to the server at this stage could save the password data and sell it to third parties).

It is also worth considering that the user password's hash value is stored on the server. If the salt value is not used, picking up this value through the dictionary attack wouldn't cause difficulties for an attacker. Even if the salt value

is used, then most commonly, it is specified by the server itself and can be stolen. In this case, an attack with a dictionary will take longer, but it can still be carried out.

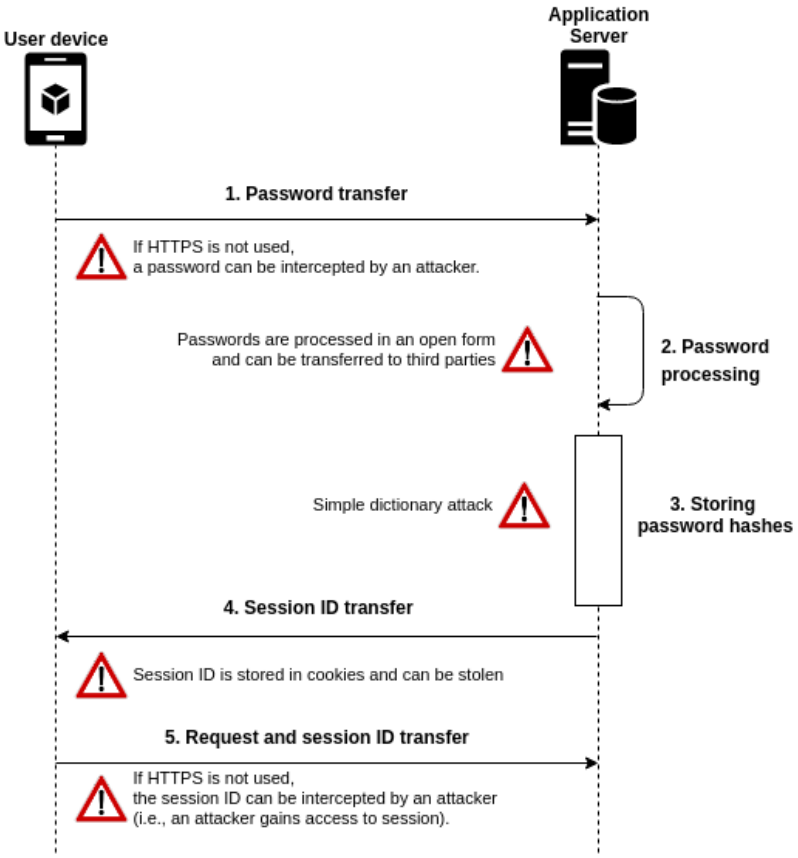

Figure 8.1 — Vulnerabilities of traditional approaches

Also vulnerability is present at the stage when server sends a session identifier value to a user. Using the obtained identifier, a user can obtain the server's response without having to re-authenticate for each new message to the server. This identifier is stored in the user's browser (cookies) and can be stolen by other members of the local network (as well as by other users on the same device or by using malware).

When requested by an application server, a user sends the value of the session identifier. Note that if the transmission channel is not secured, the session identifier can be intercepted by an attacker. Having obtained the session identifier, an attacker gains access to the resource.

Building a secure storage of private user data with the ability to quickly access it has always been a difficult challenge. Many users store their data locally. But storing a private data on a local device comes with a number of risks such as the loss or theft of the device. In this case, if the data were in a single copy, then a user can ultimately lose access to them.

Using the password to encrypt the key data

Storing the key data on a remote server allows a user to conveniently access their keys at any time and from any device that has an appropriate software. It is important that the key data is stored in an encrypted form, and encryption must be performed directly by a user. Also, in the event of an attack on the key server, the key data won't be compromised if the secret for decryption is known only to the user.

To ensure these properties, it is assumed that before sending the key data to the service, a user will put them in a secure container (encrypt) and transfer this container to a server for the storage. In this case, the server will not receive any information about the data stored in the container (Figure 8.2).

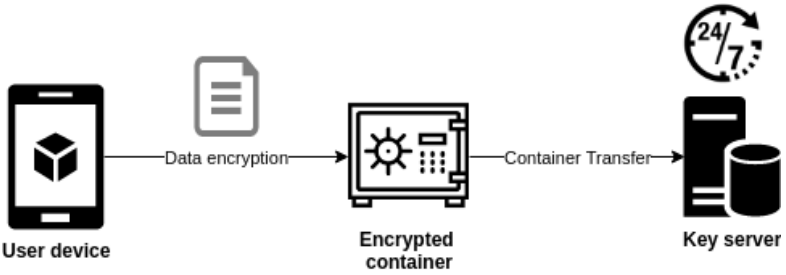

Figure 8.2 — Procedure of encrypting key data and sending it to the server

The figure provides the scheme for the generation of the encryption key from the user's secret (password). For this, the one-way key derivation function (KDF) must be used which takes a set of parameters at its input [39].

Cipher Key = KDF (user ID, salt, KDF-parameters, password) where

- *user ID* is determined by a user and serves as a user identifier in the system (for example, an email address)
- *salt* is generated by a user during the registration and is stored by the server
- *KDF-parameters* are determined by the server and contain parameters of the used one-way function (function identifier, output size, etc.)
- *password* is a secret value that is determined by a user and processed only on the user's device

Thus, only parties possessing a secret (password) can decrypt the container and gain access to the key data. As to the encryption algorithm, a symmetric algorithm in the block integrity verification mode should be used (GSM). The described approach allows ensuring the integrity and confidentiality of the key data stored by the client as well as avoiding the possibility of exposing the keys even by the service staff.

Object registration procedure

To understand what kind of data a server can operate with to authenticate an object, consider the process of registration (it also includes the process of the container creation which is discussed above). The registration process is shown in Figure 8.3.

The client registration process consists of the following steps:

1. Object sends the registration request to the server.
2. Server sends KDF-parameters to an object.
3. Object generates the salt value and generates the container encryption key (see *Using the password to encrypt the key data* subsection).
4. Object encrypts the key data (i.e., creates a container).
5. Object forms the Wallet ID value (process is described below).
6. Object sends an encrypted container and the following values to the server: user ID, salt, KDF-parameters, and Wallet ID.
7. Server saves the received data.

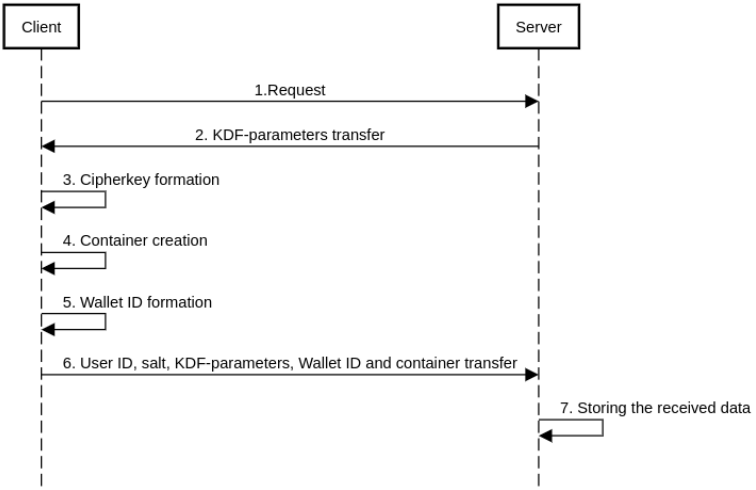

Figure 8.3 — Object registration process

Note that when an object is being registered and authenticated, confidentiality and integrity of the data transmitted to the server must be ensured. To do this, you can either use the one-directional encryption on the service key and use the hash function or establish a shared secret between objects and use the symmetric encryption with Message Authentication Code (MAC).

The *Wallet ID* value is generated based on the value of the container's encryption key. To generate the *Wallet ID* value, it is recommended to use a cryptographic hash function.

$$Wallet\ ID = Hash(\text{“}Wallet\_ID\text{”} \mid\mid Cipher\ Key) \text{ where}$$

- *Cipher Key* is a container encryption key
- *“Wallet_ID”* is a constant string set by the service

Note that during the registration, a secret (password) of an object is processed only on their device and is never stored by the server (unlike the CHAP and Kerberos protocols [40; 41]), and therefore it cannot be stolen while being transmitted through the network (because it is not) or due to an attack on the server. Reliability of the secret storage is the user’s full responsibility.

It is also recommended to use an additional factor for the confirmation of registration (via phone number or email).

Object authentication procedure

To obtain the container with keys, client must pass an authentication process (Figure 8.4). This scheme provides verification of an object's authenticity by verifying the provided identifier.

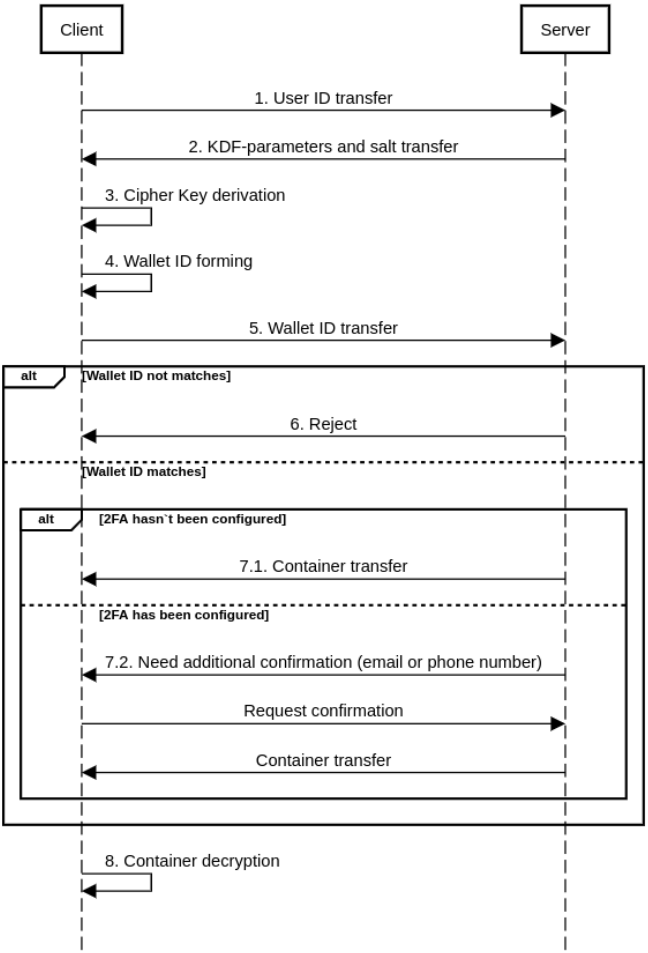

Figure 8.4 — Object authentication procedure

Note that during the authentication, the user's secret is not leaving their device. Even if an attacker steals the Wallet ID of an object, they can gain access to the container (authenticate instead of the target object), but they cannot decrypt it as this still requires to know the user's secret.

Key server security model

Keys server can be used separately from an application server and perform user authentication with the subsequent transfer of a container with the user private keys. It is enough for an application server to store user public keys whereby verify the signatures in the received requests. Such an interaction proves to be more secure since the user secret data is never transferred through the network (Figure 8.5).

To gain access to an application service, a user must get the container from the key server. To do this, she derives the container encryption key and the *Wallet ID* value from her password. Note that at this stage, it is important to check that the software does not have backdoors that could leak the encryption key or password.

A user sends the *Wallet ID* value to a key server. Even if the transmission channel is not secured, no data that would compromise user passwords or other sensitive data is transferred (an interception of the *Wallet ID* value would, at best, only allow an attacker to obtain a container which is encrypted, and which only a user knows how to decrypt). The key server transfers the container to the user.

Thereafter user receives the container and decrypts it using their password. Noteworthy, at this stage, it is important that the user's software doesn't have any vulnerabilities, which would allow an attacker to intercept the user password while it's being processed locally on the device.

The user decrypts his keys and accesses an application server. At the same time, he signs the request with his private key. Even if an attacker manages to intercept the request, they wouldn't be able to change it, or extract some information that would compromise the user's private key or password from it. Also, an attacker wouldn't be able to reuse this request since it includes a timestamp.

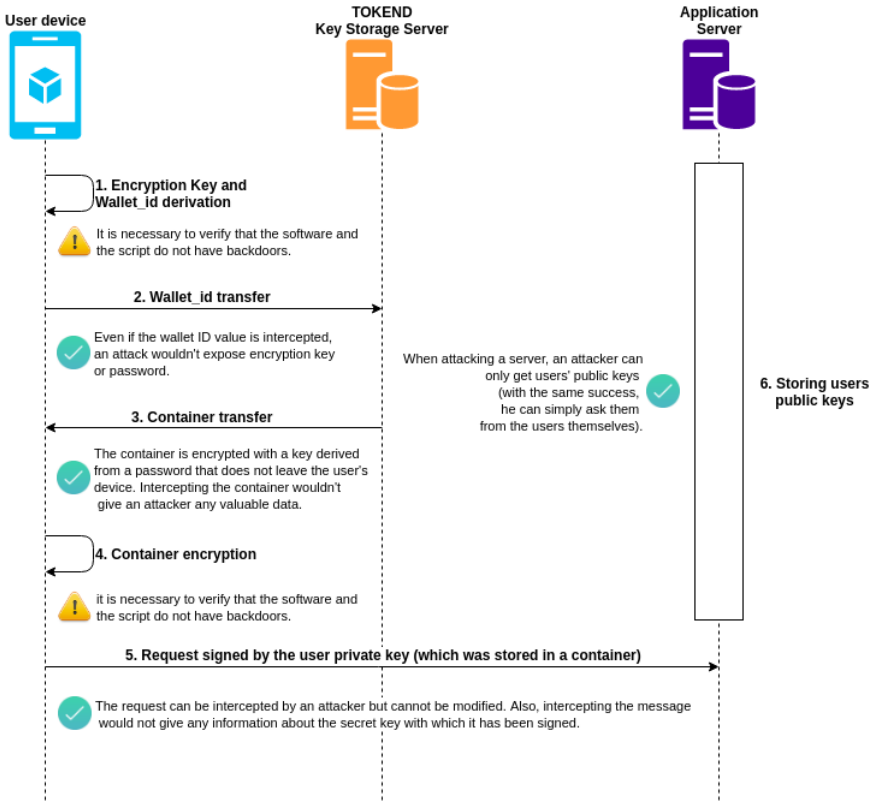

Figure 8.5 — Accessing the application server

An application server stores the user public keys all this time. After receiving the request, it verifies the signature using one of the stored public keys. Providing an attack on the application server does not make sense since public keys, by definition, are publicly available and having them, it is practically impossible to compromise the private keys.

Security assumption and integration with SSO systems

Described approach removes the responsibility of the server owner to store user passwords and encryption keys. Confidentiality of the user data is directly dependent on the users and the reliability of their passwords.

- An attacker cannot steal user passwords from the server

- The server owner will not be able to sell passwords even if he wants to
- Nobody but a user knows what is inside an encrypted container (e.g., private keys, documents, photos, etc.)
- An attacker may steal or intercept the *Wallet ID* value, but they will not be able to recover an encryption key and password
- Having the server hacked, an attacker can steal user containers, but this wouldn't present real value for them since to gain access to user accounts, their passwords are required
- A server may fail. Note that in this case the data wouldn't be compromised, but a user will lose access to it for a while. However, this problem can be solved by backing up the data or by using several independent key servers to store the container

Key server can be integrated with SSO systems. The main task of such systems is to create a user-friendly working environment for authentication while ensuring a high level of protection. SSO technology involves the installation of a client (agent) on a user's workstation that automatically fills in authentication forms with user sensitive data. That is, a user needs to go through the authentication procedure only once, and all the other authentication processes will be performed directly by the SSO agent.

Key server allows storing user authentication data for application servers access in an encrypted container. After authentication to the key server, a user receives the container, decrypts it, and passes the contents of the SSO to the agent. All further actions related to the authentication will be performed directly by the agent.

9 FUNCTIONALITY OF THE ACCOUNTING SYSTEM'S CLIENT

To begin with, let's consider the basic functions that must be implemented by client software:

- ❖ *Creating and managing accounts*
- ❖ *Generating and processing keys*
- ❖ *Retrieving information on accounts and balances*
- ❖ *Signing requests and transactions (and sending them)*
- ❖ *Storing all confirmed transactions associated with a particular account*
- ❖ *Storing keys (optionally)*

The client software must be able to create and sign transactions as well as send them to the necessary accounting system. The necessary level of protection when generating and processing keys must necessarily be provided. A client application can also provide the ability to store keys locally on the client's device in a protected form.

Another requirement is the ability to provide information about users' balances and accounts. Also, the client software must store all the transactions associated with the user in order to reliably prove that a particular transaction has actually taken place at a particular moment in time (this may help in the case of any disputes).

In this section, we will observe the basic principles of how users authorize to the accounting system and what is the role that the client software performs when authenticating. In the next two sections, we will consider two important functions of an accounting system: crowdfunding and voting—as well as security and privacy issues related to these functions.

9.1 Authorization principles

The best practice in client-server communication in terms of security is to sign each request and response

Issues with sessions

The general principle of a session is that a user provides her password and after the successful authentication, a session with this user is opened, and the user receives the session identifier. Subsequently, a user will stay within the session to receive the requested data by attaching the identifier to each message that is sent to the server.

The session identifier is stored by both the client (in cookies) and the server (in the sessions database). Note that sessions are managed by the server exclusively: the server generates the session identifier (initiates the opening of a session) and decides when the session should be closed. Thus, for every user who interacts with the server, a separate entry in the sessions database is created (this record exists in the sessions database after the session is opened and before it is closed). Let's look at the basic steps of how sessions operate in order to analyze at which stages vulnerabilities arise (Figure 9.1).

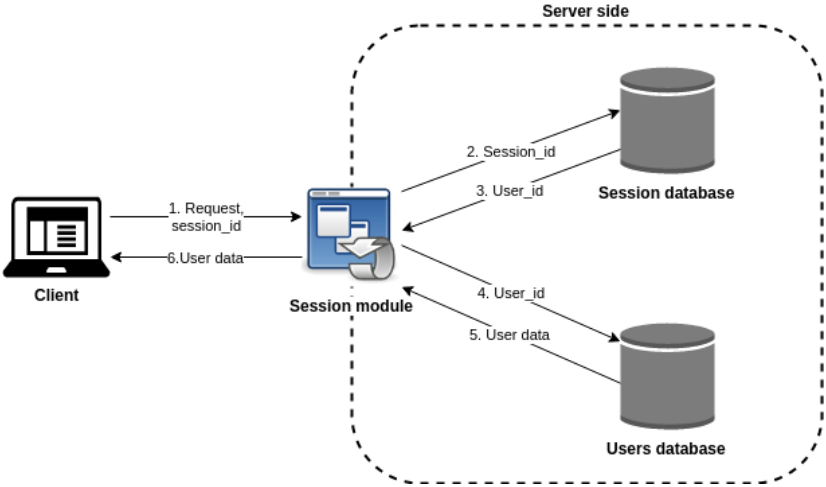

Figure 9.1 — Sessions mechanism

During the client authentication, the server checks the client's identifier and password, generates the session identifier, and saves the *session_id* and *user_id* values into its sessions database. Next, the server sends the session identifier to the client, and the session between the server and the client is opened. As a result of these actions, the user can receive the information from the server without having to re-authenticate for each new request sent.

A potential threat with sessions is that an attacker can intercept the client's message, obtain the identifier and thus gain access to the server by pretending to be the target client.

Session hijacking

When authentication is successfully completed, the session starts and remains active until the end of communication (the timeframe is determined by the server). Hijacking a session presumes that an attacker invades and takes advantage of an open session. This intrusion can be either detected or not [42].

By launching such an attack, the attacker essentially exploits the client's session identifier (which is stored in the client's cookies) and gains access to the services provided for the target client.

The prerequisite of a session hijacking attack may be the fact that the server isn't responding to the client's input as expected, or it completely fails to respond for any reason unknown.

The basic approaches for implementing the attack are the following:

- Session sniffing [43];
- Cross-site scripting [44].

Protections against such types of attack exist, however, since the security mechanisms are quite complex, not all services use them because they decrease the level of convenience for the client.

Usage of signatures for requests to interact with the server

To avoid this type of attack during the client-server interaction, the server must authenticate each message sent to it. In doing so, it is critical that the message is not modified by a third party while transmitted. Therefore, modern systems do not use the sessions mechanism but rather an approach that verifies

every single message (both the client's requests and server's responses) in terms of its authenticity and integrity. This is possible by signing each message with the corresponding key (Figure 9.2).

The digital signature mechanism is widely used to verify the authenticity and integrity of messages. Notably, this approach increases the security of the client-server interaction without complicating the entire process—no sessions are opened. The server only needs to have the client's public key whereby it can explicitly authenticate each message that the client signs with the corresponding private key.

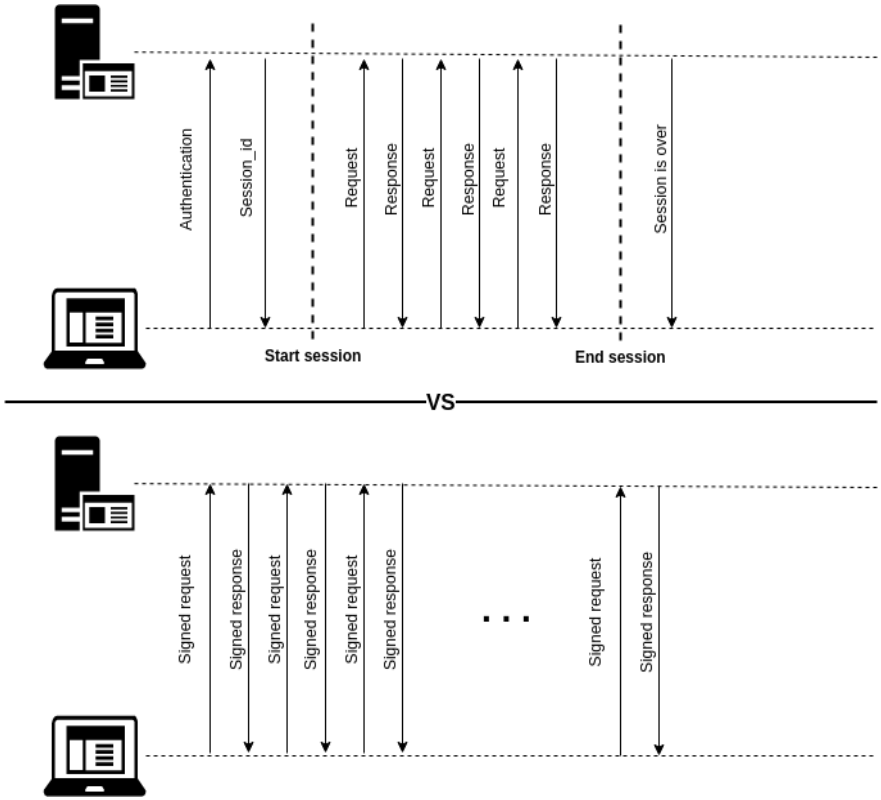

Figure 9.2 — Sessions versus signed requests

Advantages of the described approach

- ❖ *Access is gained only by the key owner*
- ❖ *Attacks exploiting vulnerabilities arising from the sessions mechanism are excluded*
- ❖ *Integrity and authenticity of every request is ensured*

9.2 Crowdfunding

Crowdfunding is one of the important steps towards the decentralization of decision-making and democratization of investment

Having a complete set of tools to manage the permissions of investors and fundraisers which are based on the identification data obtained from them, is of paramount importance.

To begin with, we define the roles that are present on the crowdfunding platform.

- ❖ *Asset owner*
- ❖ *Investor*
- ❖ *Crowdfunding administrator*

The asset owner has the right to create crowdfunding campaigns and sell tokenized assets to investors. An investor is a role that has the right to participate in crowdfunding campaigns. To enhance the security of crowdfunding, a request must be processed by the administrator with appropriate permissions. The crowdfunding process must obtain the following features:

- ❖ *Assets issued and investors' deposits are locked and distributed via the smart contract functionality*
- ❖ *Investors can at any time cancel their investments (before the end of sale)*
- ❖ *Possibility to set the minimum and maximum amount of expected investments*

The first feature means that the campaign initiator doesn't have a technical

possibility to outwit his investors. After the campaign is created, the corresponding assets issued, and funds invested, they should be locked, and the initiator shouldn't have access to them until the end of the campaign.

Investors must be able to reinvest or cancel their investments at any time after the start of the campaign and before its end.

Correspondingly, there should be a possibility to set up the minimum and maximum amount of investments allowed by the campaign initiator. If the investment amount exceeds the hardcap, the campaign ends immediately, and the assets are distributed to the investors. Similarly, assets are distributed if the softcap is reached but in this case, only after the campaign time is over. And finally, if the campaign end time is reached but neither maximum nor minimal expected investments amounts are reached, then the campaign is considered failed, and all invested funds are distributed back to the investors.

Crowdfunding process

To start their own crowdfunding campaign, the fundraiser should have the appropriate permission, which is issued by an administrator (with appropriate rights) who confirmed her KYC request. The KYC procedure can both be performed internally in the system as well as delegated to the external provider (the integration principles will be described in section 10.4).

Having obtained the needed permissions, the user can create and pre-issue tokenized assets that will be put up for sale. Next, the user creates a crowdfunding campaign request, which also has to be approved by the administrator with the required rights.

Note that investors must initially pass an appropriate identification procedure, which will allow them to get the right to accept an asset and trade it on the secondary market.

The module that is responsible for crowdfunding has the following features:

- *Whitelisting accounts*
- *Locking and unlocking funds, depending on the conditions of the smart contract*
- *The function of calculating shares and dividends, depending on the contract execution results*
- *Ability to limit accounts in terms of asset trading and transferring if needed*

The process of the crowdfunding campaign is shown in Figure 9.3.

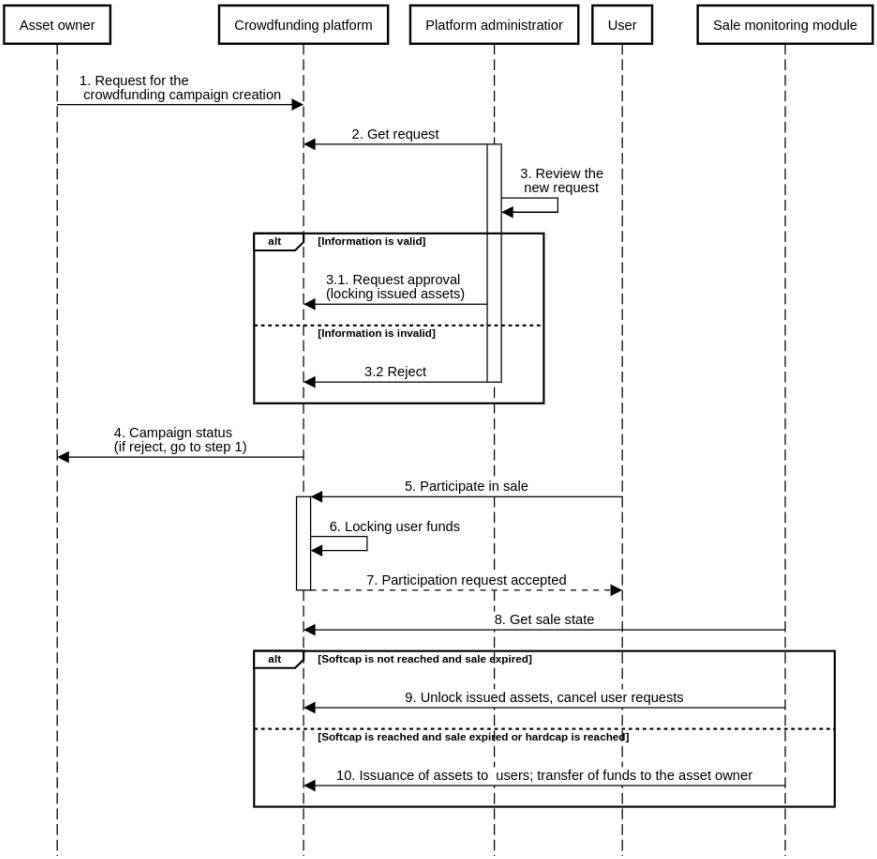

Figure 9.3 — Crowdfunding campaign flow

1. Asset owner account creates a request to conduct a crowdfunding campaign through a smart contact and sends it to the platform. The created request indicates the following details: the date of the campaign start and end time, the minimal and maximal expected investment amount, the asset (or the set of assets) to be issued; the assets/fiat currencies that users will be able to invest in the campaign initiator's tokenized asset; the roadmap for investors that shows how the raised capital will be used.

2. Platform administrator account (with permission to confirm crowdfunding campaigns) applies to the platform to view all the unprocessed requests.
3. Administrator reviews the campaign details.
   3.1. If the administrator agrees with the campaign details, he accepts the request. Note that the confirmation may require signatures from multiple administrator accounts. The corresponding transaction is created.
   3.2. If the administrator does not agree with the details, he rejects the request. The refusal must indicate the rejection reason.
4. Asset owner account receives the result of his campaign creation request. If the request is denied and she wants to try again, the account starts over from Step 1. If the request is accepted, then the crowdfunding campaign will start at the time specified in the smart contract.
5. Investors are notified about the new crowdfunding campaign on the platform and can start investing the allowed assets (or fiat currencies)
6. Investments are locked via the smart contract so that the asset owner account can access them only if the campaign is successful (i.e., if the required conditions of the contract are successfully fulfilled). Investors can cancel their investments (as well as reinvest them in other projects).
7. After the funds are locked by the smart contract, investors are notified about their participation in the campaign.
8. Sales monitoring module constantly checks the crowdfunding campaign state.
9. If the campaign time is over and the minimal expected investment amount isn't reached (i.e., the campaign failed), then all the collected investments are distributed back to the investors' accounts, and the corresponding asset is unlocked for its asset owner account.
10. The crowdfunding campaign can be considered successful in two cases.
    10.1. The first implies that the investment amount reached the maximum expected investment amount. In this case, the

crowdfunding campaign is finished immediately without waiting for the expiration date. Invested funds are transferred to the asset owner account, and the released tokenized asset is transferred to the investors' accounts.

10.2. In the second case, both the minimal expected investment amount and the campaign expiration date should be reached. Note that investors cannot cancel their investment after the campaign ends successfully.

Secondary market for trading assets

An important requirement for a crowdfunding platform is the possibility to trade assets purchased on the secondary market. Notably, one of the non-trivial and crucial requirements here is the ability to regulate—it is much more difficult to control the secondary trade by multiple investors than to control a single campaign initiator. Therefore, as we mentioned in section 3, in the accounting system, it is necessary to ensure the differentiation of access permissions, namely the ability to create administrator accounts that can change permissions for other groups of accounts (through cryptographically signed transactions).

9.3 E-voting functionality

Getting rid of non-transparent counting committee and truly secure decision-making are the consequent results of applying cryptography + consensus

Traditional voting systems are no longer effective in terms of their requirements: a host of printed voting papers, pseudo-anonymity of voters, non-transparency of voting counts, dependence of the (entire) voting procedure on a central authority. In fact, these are only the most critical of the issues that exist in the current voting systems.

In recent years, digitizing the voting process has gained increasing momentum. The most striking examples are the implementation of a digital voting system for electing local authorities in Estonia (which started as far back as in 2005 [45]) and the attempts to introduce such a system in Switzerland.

However, existing solutions still have a number of drawbacks, in particular, vulnerabilities associated with a central authority validating all results.

In this section, we will describe how the accounting system can be transformed into the voting platform while maintaining its participants' anonymity given that the calculation results are fully transparent.

Components and structure of an e-voting system

The decentralized system of anonymous voting consists of but is not limited to the following elements:

- ❖ *Validator nodes*
- ❖ *Auditor nodes*
- ❖ *Identification system*
- ❖ *Users (voters)*

Schematically, the hierarchy of components and their relationship is presented in Figure 9.4.

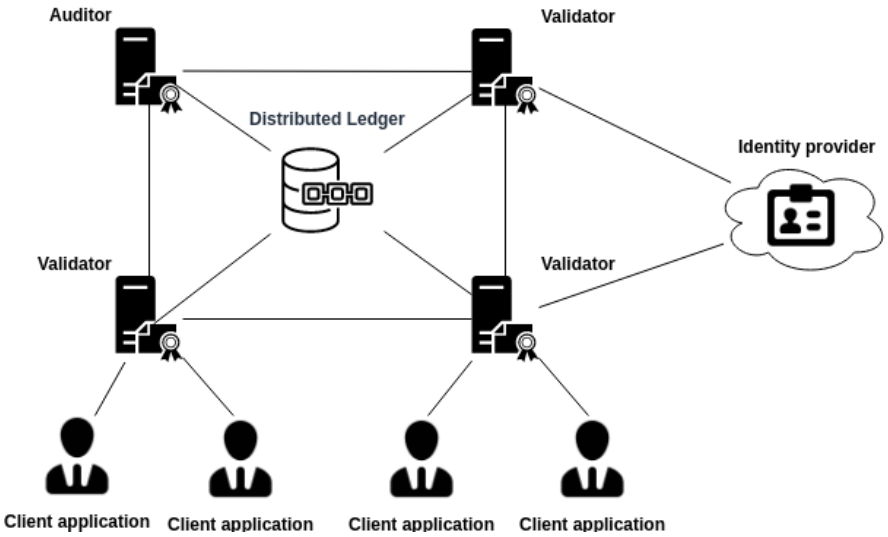

Figure 9.4 — Components of a decentralized e-voting system

The purpose of all the components depicted above was described in section 2.1. Accounts perform the role of voters in the system. They independently vote for making a certain decision. Each vote is a transaction signed by the user directly.

Voting process

In order to vote, a user must create and sign an appropriate transaction. The transaction structure may be as follows:

Transaction id
Nonce
Candidate id
Timestamp
Signature

Transaction id is the hash of all other transaction fields. *Nonce* contains a random value and is used to make the transaction unique. *Candidate id* contains an identifier of the voting entity for which the voter wants to to give their voice.

Timestamp is the UNIX-style value of the transaction creation time. *Signature* is the transaction digital signature value.

The voting process consists of the following steps (Figure 9.5):

1. User creates a transaction and signs it. Then he sends the signed transaction to the validator. The validator verifies whether the basic transaction fields are valid.
2. Validator applies to the identity source to check if the user with the corresponding public key has permission to vote.
3. If the corresponding public key has the necessary permissions, the validator (or a group of validators) accepts the transaction and updates the ledger state. Otherwise, the transaction is rejected. After that, the

user receives proof that his vote was taken into account signed by the platform validator(s).

4. User can make sure that her voice has been accounted for properly. In case the platform provides for such a possibility, then the user can have a full auditor node (that stores the entire transaction history) and be able to verify the votes herself, or if the platform doesn't allow for that, then she can apply to one or several full nodes she partially or fully trusts.

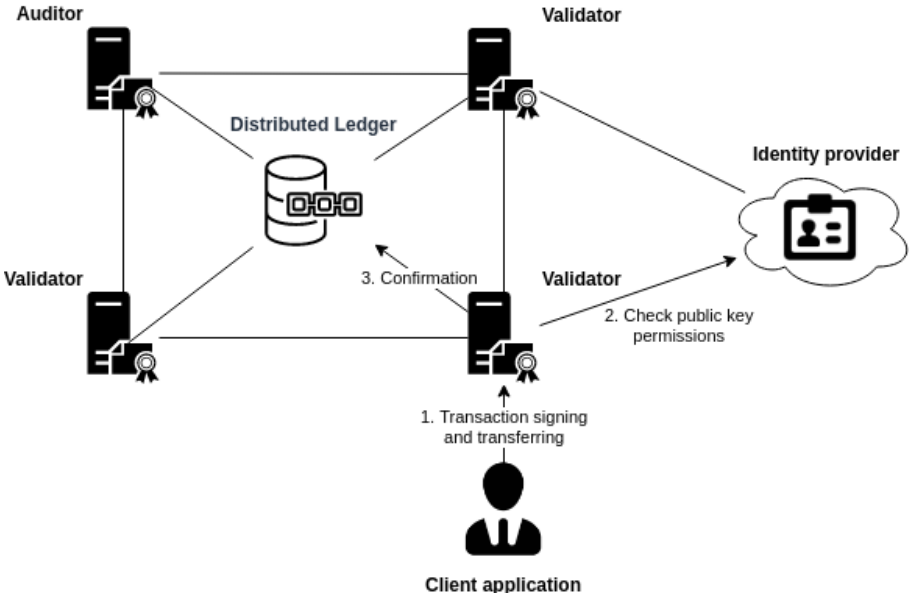

Figure 9.5 — Voting process

Anonymous e-voting using ring signatures

The approach described above enables e-voting while ensuring the transparency of processes and the integrity of the voting history. However, some voting systems also require another property for system users, namely, anonymity. Further, we will describe how to ensure voters' anonymity while maintaining all other properties, namely transparency and integrity, of the accounting system.

Ring signature principles

Ring signatures are used to ensure the anonymity of individual users in a certain set of members of a group (a ring) [46]. To generate such a signature, a user needs other users' public keys and his own key pair. During the signature verification, a verifier can check that it was generated by one of the ring members, but she cannot determine by whom exactly.

Imagine a group of *n* users, as shown in Figure 9.6. Each user has his own key pair: a private key (*SK*) and a public key (*PK*). All members can see each others' public keys, while the private keys are only available to their owners.

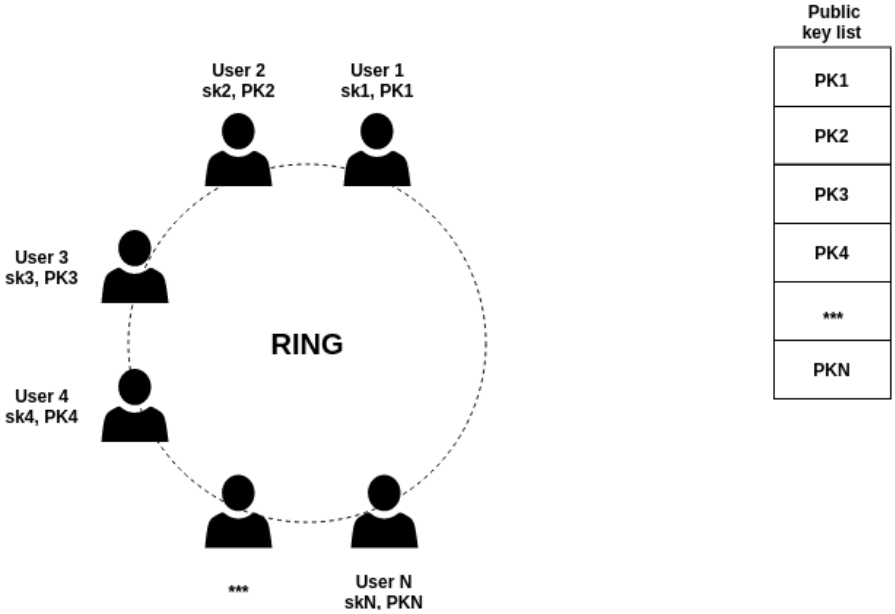

Figure 9.6 — Ring forming process

In order to generate a signature on behalf of a group, a user must input the public keys of all ring participants (including his own) to an algorithm and use his own private key as a secret value. Figure 9.7 depicts the process of how the user (User 4) generates the ring signature.

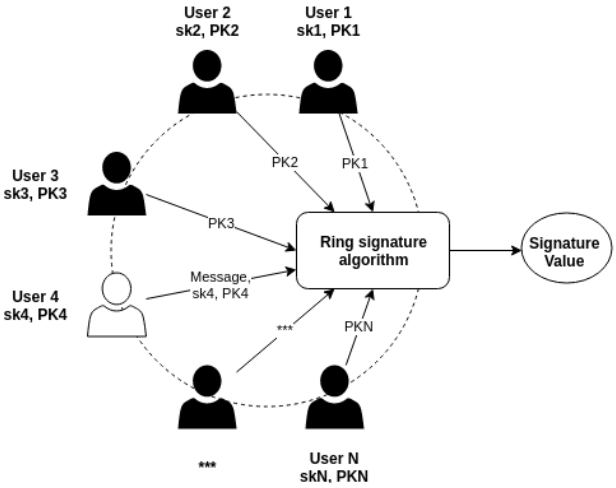

Figure 9.7 — Ring signature generation

When the verifier inspects the signature value, he can make sure that the signature was generated by one of the group members; yet he cannot find out by whom exactly (in fact he can, theoretically but only with the $\frac{1}{n}$ probability). Noteworthy, if all group members collude, a user can be disclosed (Figure 9.8).

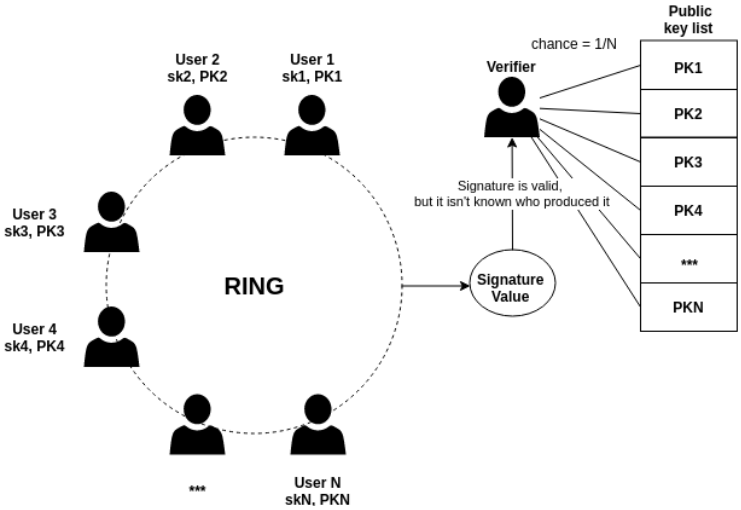

Figure 9.8 — Signature verification process

Anonymous voting

The transaction structure in an anonymous voting system is almost the same as the one described above, but it contains one extra field: *public keys of the group*.

Public keys of the group is a list of the ring participants' public keys (those used to generate the signature). It also contains the voter's public key, but the position of this key is unknown.

In order to sign a transaction and ensure the anonymity of a vote, a user selects the list of other users' keys. It is important that the selected public keys actually belong to the other voters (i.e., that they have permission to vote). The list of public keys of voters should be open to all system participants. This list is created before the voting starts (once you are registered and provided your public key, you have been included in the voters list).

The number of selected keys depends on the voter's anonymity level. If the selected group is small, then the probability to deanonymize the voter is much higher.

After a user has selected a set of public keys, he calculates the ring signature value for the transaction and then sends it to one or several platform validators (Figure 9.9).

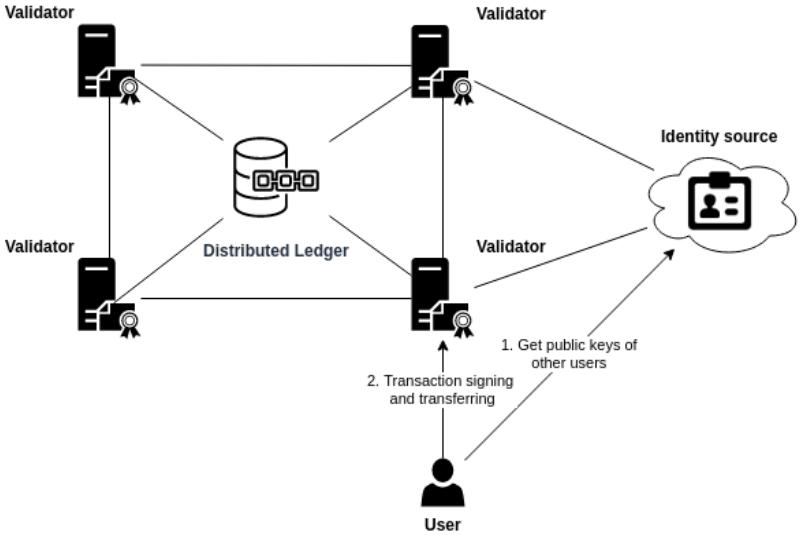

Figure 9.9 — Anonymous voting

After a validator receives a transaction, she must verify that the sender has the right to vote. Note that the validator does not know the transaction sender's identifier (more specifically, she does not know which of the public keys specified in the transaction belongs to the voter). Therefore, she needs to verify permissions of all keys specified in the transaction.

If all the specified keys include the voting permission, then the transaction is correct and can be confirmed. At this stage, the validator must verify that the user cannot create and send several transactions from different groups—otherwise an attacker will have an opportunity to create several transactions by constantly changing groups, and these transactions will be considered valid; all because the sender of each transaction is unknown. In such a case, the image of the private key must be used as a security mechanism. Since the image is unique for each key pair (and it is used for creating and verifying signatures), a user cannot sign several transactions using the same private key. The transaction confirmation process is shown in Figure 9.10.

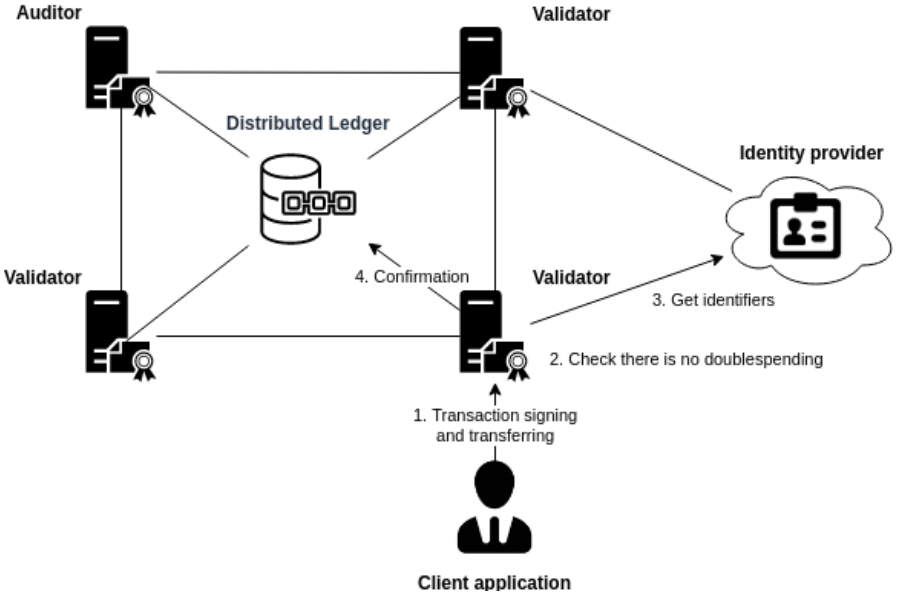

Figure 9.10 — Transaction confirmation process

Features of a decentralized e-voting system

The described approach includes the following advantages:

- ❖ *Possibility to verify the voters' permissions*
- ❖ *Anonymity*
- ❖ *Possibility to verify whether a particular vote is accounted for correctly*
- ❖ *Inability to perform the double spending attack*

On the one hand, this approach allows validators to verify that a transaction sender has permission to vote (if he used existing public keys of other participants).

At the same time, who of the group has voted can only be determined with a certain probability by the validators (the bigger is the ring size, the lower is the probability). Also, a user can be fully deanonymized if all other group members collude (and disclose their own votes).

Every user can make sure that his voice has been added to the distributed ledger. In addition, every full node owner can verify that the voting results match the set of performed transactions.

A user cannot create new transactions with different groups if a security mechanism against double spending attacks is used (the private key image).

Also, based on this scheme, a user can be allowed to change the value of her vote. In this case, it is not just a single transaction that will be taken into account, but rather the last one which was added to the chain of blocks. However, in such a case, security measures have to be designed and implemented to prevent spam attacks and other attacks that may affect system performance as well as the data stored in the chain.

10 PRINCIPLES OF INTEROPERABILITY BETWEEN SYSTEMS

Interoperability of systems should be based on cryptographic proofs and identities and not on the knowledge of internal processes or the on system architecture

Principles of interoperability between systems:

1. Accounting systems that contain information about tokenized assets may not interact or even know about each other.
2. When tokenized assets are exchanged, they are not transferred between systems, but rather their owner is changed within the same system (Figure 10.1).

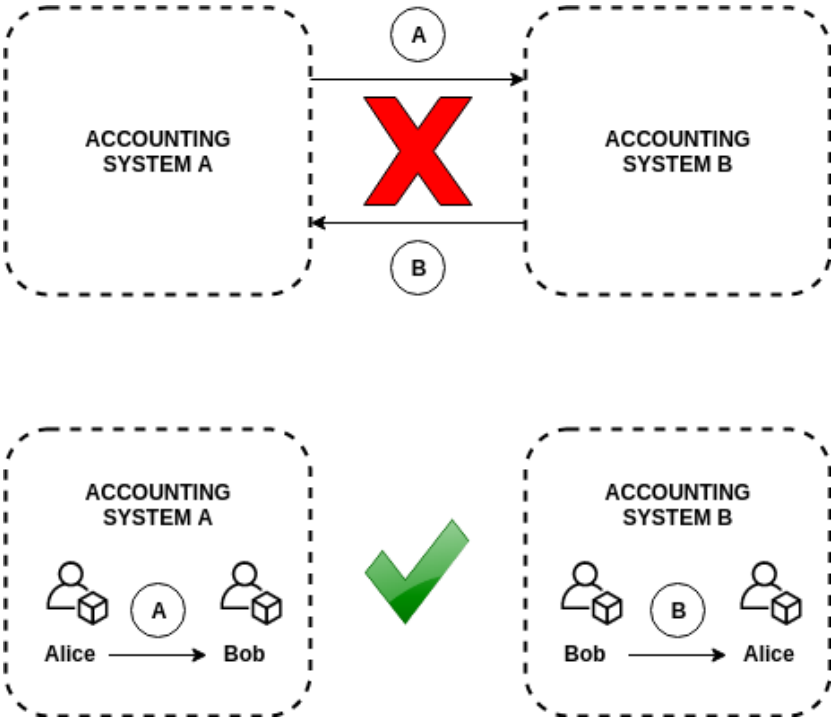

Figure 10.1 — Assets do not leave the boundaries of the accounting system

3. User identifiers can be reused across systems, thus, there should be no need to ask for it every time (Figure 10.2).

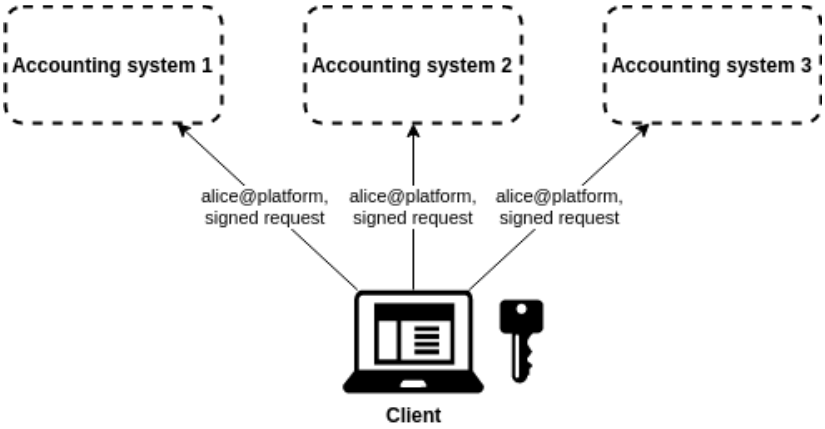

Figure 10.2 — One identifier for all accounting systems

4. To ensure that all required parties agree on the conditions of a transaction, multisignature is used (Figure 10.3).

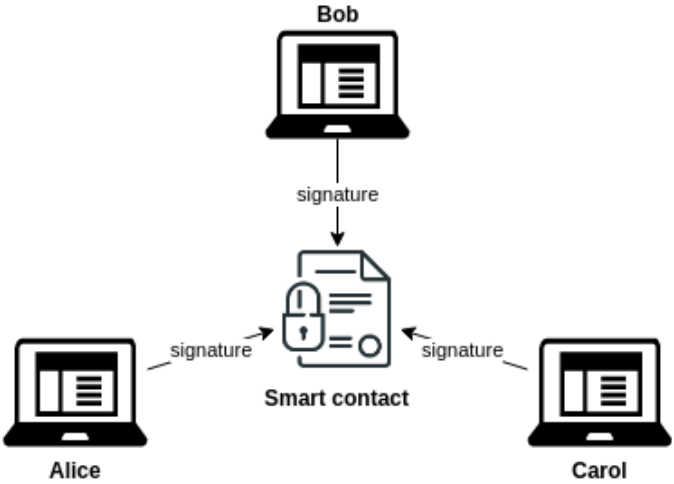

Figure 10.3 — The contract is signed by all parties involved

5. The only thing that matters: the prepared transaction "knows" who must sign it in order to make it valid.
6. Users explicitly see the conditions that they are signing.
7. There is no need to monitor whether consensus is reached in another system.
8. Aggregating signatures make the resulting signature compact while keeping the signers private.
9. The public key of an aggregated signature is known beforehand and can be placed in the transaction structure prior to executing a transaction.
10. The transaction validation process is asynchronous (there is no guarantee that two transactions will be validated simultaneously or even validated at all) because of network delays, insufficient fees, or AML requirements. However, this can be solved by retrieving all the approvals/signatures from the corresponding modules before the transactions are signed by the participants. The module's signature may have an expiration date, for example, in the case of potential fee modifications.

10.1 Multisignature approach

Multisignature allows distributing the management of actions or processes between parties that may not trust each other

Despite the common opinion, when the assets of two accounting systems are exchanged, these systems do not interact with each other. A tokenized asset **never leaves** the boundaries of its accounting system. As we mentioned earlier, all that happens during the exchange, which is the change of asset owners within the systems that account for these assets.

Usage of the 2-of-2 multisignature and uniform accounting systems

If accounting systems are similar to each other—they support a uniform transaction structure, have common digital signature algorithms applied—then

using 2-of-2 multisignatures allows a trustless exchange between the participants of the respective accounting systems.

To perform the trustless exchange, parties create a uniform transaction that requires 2-of-2 signatures to be validated. The transaction consists of a contract and sets of signatures. To create a contract, parties build a Merkle tree of their transfer operations. The root of the Merkle tree is placed in the contract body. Further, the transaction is signed by both parties and propagated to both accounting systems. In each accounting system, only the operation related to this system and the Merkle branch, proving that the operation is included in the contract body, are published.

Thus, each accounting system may not be aware that the contract contains the operation that relates to the other system. What validators see is just a transaction containing several conditions with only one of them fulfilled (Figure 10.4).

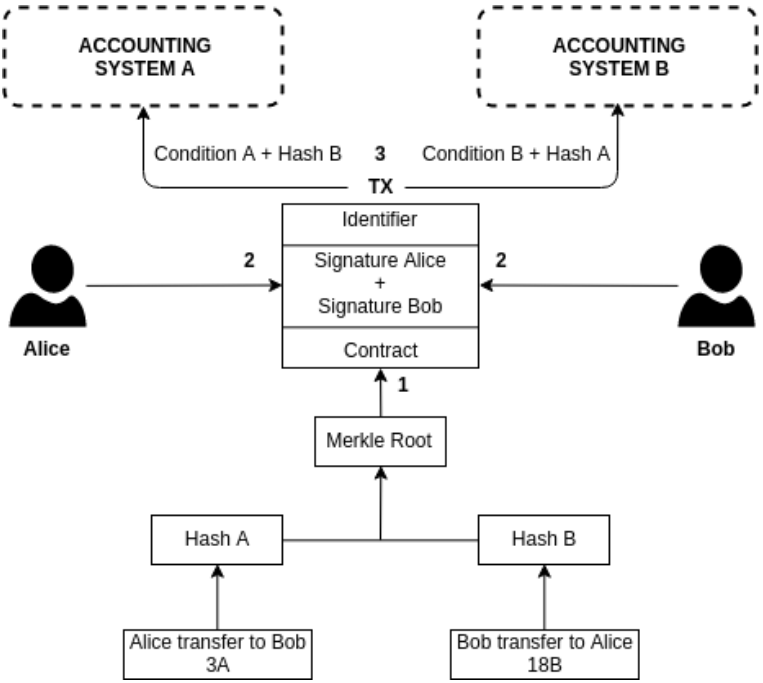

Figure 10.4 — 2-of-2 multisignature

Usage of 2-of-3 multisignature and a trusted intermediary

Trusted third parties were traditionally used for online trading between two parties. They were supposed to ensure that transactions are executed correctly and the parties fulfill their obligations. In fact, such entities require the transacting parties to fully trust each other since they can (technically) take all the funds away (Figure 10.5).

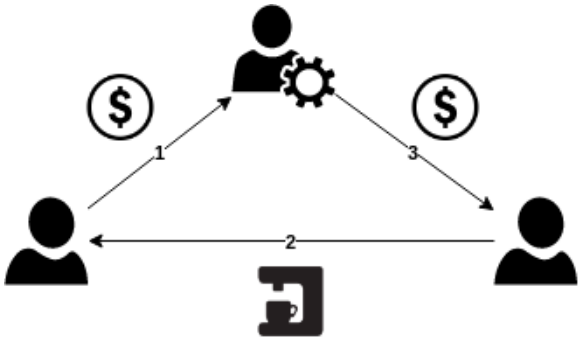

Figure 10.5 — Fully trusted mediator

The multisignature mechanism is the solution that eliminates the need for trust. As in the previous case, there are three parties involved: two transaction parties and a mediator. In this case, however, after the funds are locked by the counterparties, unlocking them requires 2-of-3 signatures (two of which belong to the two counterparties and one to the mediator). In such a case, the mediator cannot steal the funds (unless he colludes with one of the counterparties). More on that, if transaction parties have successfully agreed with each other, then the trusted third party's signature is not even needed. This approach is often used for transactions that do not occur simultaneously (for example, delivery of a physical product): when neither the buyer nor the seller wants to take the first step, and they need a third-party to resolve the potential conflict.

At this point, the question might arise, why need this mediator? The answer is, to resolve the disputes between the two counterparties. What the transacting parties essentially do in this case is that they both provide the evidence to the mediator that is required for conflict resolution. The mediator, in turn, proposes his transaction to the counterparties which is signed by him and which

distributes their funds. If at least one of the parties is satisfied with the proposed conditions, then this party signs the transaction with the second essential signature, and the transaction is published to the network (Figure 10.6).

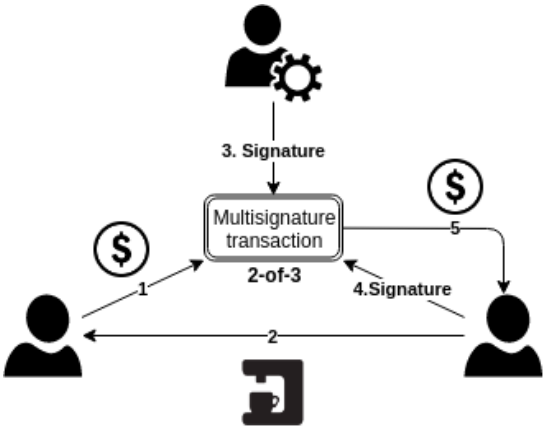

Figure 10.6 — 2-of-3 multisignature with a mediator

Similarly, assets can be exchanged between different accounting systems. In this case, both parties choose a mediator and create transactions that distribute their assets only with a multisignature provided. If the parties act honestly, then they can sign transactions and transfer assets to each other. If, however, one of the parties tries to cheat, the other one has a proof of the transfer (in the form of a transaction), the mediator can sign the required transaction and resolve the dispute.

Several trusted mediators

As mentioned earlier, there is a risk that the mediator may collude with one of the counterparties. The partial solution to this problem lies in choosing several **independent** mediators. Yet, this introduces another problem: if the number of mediators is larger than the number parties to the transaction, mediators can collude and take the counterparties' funds away, at least when typical multisignature is used.

In such a case, the solution is the threshold signature mechanism [47]. Its main difference is that the participants' keys now have weights. This means that

for the signature to be considered valid, the sum of all signature keys' weights must be greater than or equal to 1.

Let's consider an example where Alice and Bob lock their funds using a threshold signature. They set the weights of their keys as 0.5. They also include nine independent mediators to resolve the potential dispute and give each of them keys with the weight equal to 0.1. In such a case, if Alice and Bob have no conflicts and both act honestly, the interference of mediators is not needed (Alice and Bob both have keys which sum is equal to 1, and this is enough for the signature to be considered correct). If, however, say, Bob tries to cheat, then to resolve this conflict, Alice will need the "support" of at least 5 validators (Alice's keys weight are equal to 0.5, and each validator has keys equal to 0.1). Note that in such a scenario (Figure 10.7), mediators will not be able to take the funds away.

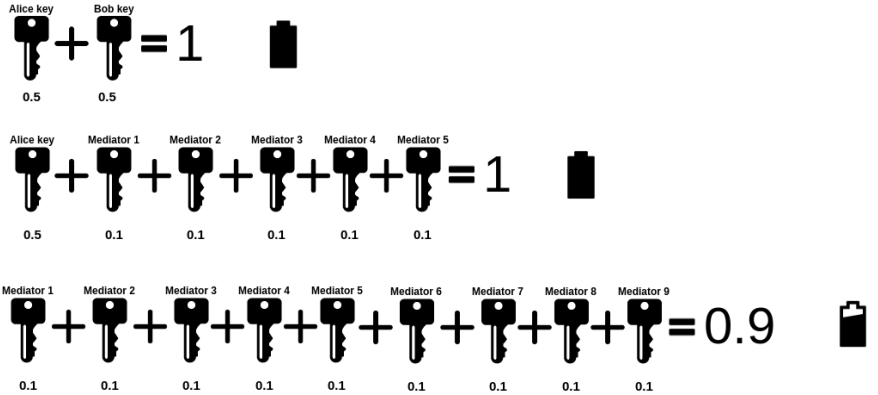

Figure 10.7 — Total mediators' keys weight is not sufficient to get a valid signature

Retrieving information from other systems

In section 10.2, we will examine in detail how assets can be exchanged between two permissionless systems without manual intervention and, accordingly, without the need to trust (atomic swaps are introduced soon).

However, in the regulated systems, this process can be organized even simpler. In such systems, the list of transactions is signed by all validators (or at least, by the majority)—in fact, this is the multisignature mechanism in its

original form. Moreover, when using signatures that allow aggregating keys (such as Schnorr signatures [48]), it is possible that each set of blocks and each transaction can be verified with the key known in advance (shared aggregated key of the platform validators).

In this case, both Alice and Bob can create 2-of-3 multisignature transactions that involve both Alice's and Bob's key and the aggregated key of one of the systems' validators (if Alice initiates the exchange, then the aggregated key of validators in her system is needed, and if Bob initiates, then in his).

If Alice and Bob are honest and have no conflicts, then they both sign the transactions and the aggregated validator's key is not needed at all. If however, one of them tries to cheat and refuses to sign the transaction with the second required key, then the other party may use the signature of validators as the second key and an atomic exchange will be performed.

10.2 Atomic swap

Atomic swap means trustless exchange between two accounting systems

What is atomic swap?

The idea of atomic swap was first proposed as an alternative to the traditional trading platforms where, when traded, digital assets are stored on the side of the platform. In essence, an atomic swap is a "swap" that is either performed inseparably (atomically) or not performed at all. It implies a trustless (without intermediaries) exchange of cryptocurrencies, digital currencies, and digitized assets [49].

To ensure the support of atomic swaps, the following fundamental requirements must be met:

- *Possibility to create a contract with a time delay*
- *Support for the same hash function by the contracts of interacting platforms*
- *Presence of an off-chain communication channel between users*

Atomic swaps use the so-called *time-limited locks* (hashed time lock contracts). In fact, these are the contracts that are supported by the accounting system itself. A pair of special contracts ensures that if the owner of one asset is changed, then the ownership of the other asset should be changed as well. And if one of the contract parties does not confirm the transfer in time, funds will be returned to the initial owners. Thus, funds in both assets are locked until all the required conditions are met and confirmed. Funds will either be exchanged atomically or not exchanged at all.

Let's consider this process in a more detailed scheme. Imagine two platforms, Platform 1 and Platform 2, that account for the digital assets and implement contract supports. There are two participants, Alice and Bob. The process of their interaction is shown in Figure 10.8.

The exchange process consists of the following steps:

1. Alice generates a random secret x. Note that if x is not random enough or it had been leaked, then Alice can lose all the assets she exchanges.
2. Alice creates transaction 1, where she sets two fund-spending conditions. The first condition states that the funds can be spent by Bob if he provides the secret value (x). According to the second condition, the transaction can be spent only if both parties provide their corresponding signature values.
3. Next, Alice creates transaction 2, in which assets from transaction 1 are sent to Alice's address / account in two hours. Note that transaction 2 requires signatures of both interacting parties to be confirmed.
4. After that, Alice signs transaction 2 and sends it to Bob.
5. Bob verifies all the transaction fields (special attention should be paid to the transaction's block time parameter). After the transaction has been verified, Bob signs it and returns it to Alice.
6. Alice publishes transaction 1 on Platform 1. As noted earlier, this transaction contains the hash of a secret generated in step 1.

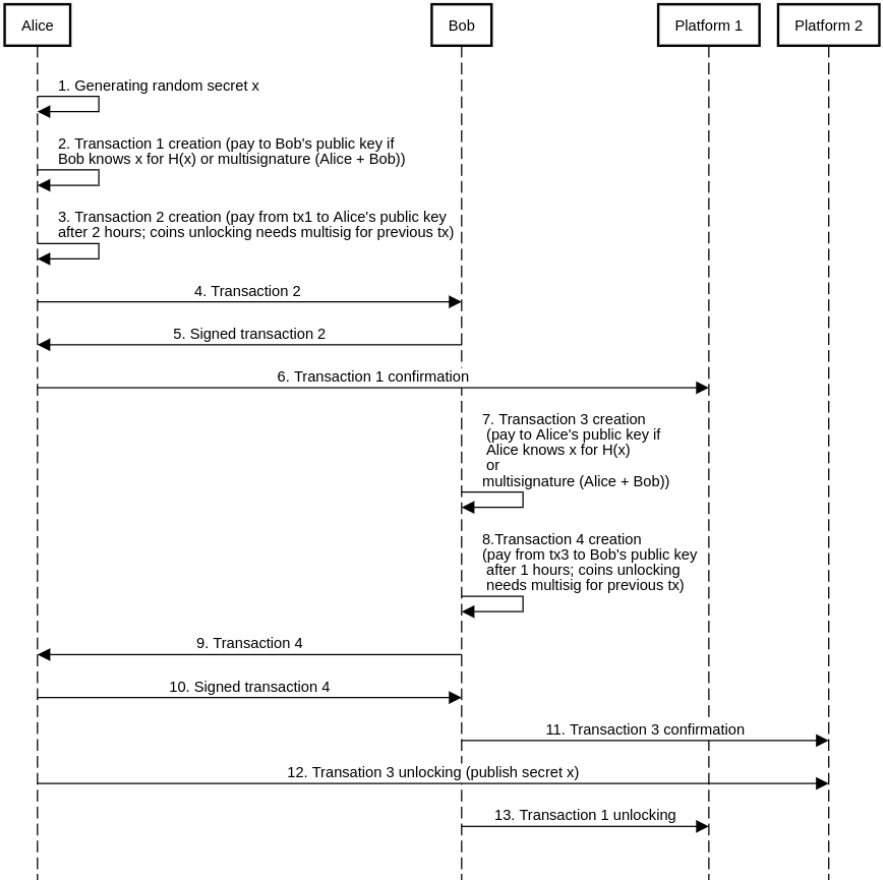

Figure 10.8 — Atomic swap workflow

7. Bob creates transaction 3, in which he also sets two conditions for spending funds. The first condition states that they can be spent by Alice only if she provides the secret (x) value (Alice knows this value as she has generated it). According to the second condition, the transaction can be spent only if both parties provide their respective signature values.
8. Next, Bob creates transaction 4, in which assets from transaction 3 are sent to Bob's address / account in one hour. Note that transaction 3

requires signatures of both interacting parties to be confirmed.

9. Bob signs transaction 4 and sends it to Alice.
10. Alice receives the transaction, verifies it, and signs it with her private key. Then she sends the signed transaction back to Bob.
11. Bob publishes transaction 3 on Platform 2. Note that transaction 3 requires a secret value generated by Alice (x) to be unlocked.
12. Alice can spend the outputs of transaction 3 only if she provides a secret value (x).
13. If Alice has provided this value, Bob can use it to spend the outputs of transaction 1 sent by Alice.

10.3 Payment system integration module

Systems that do not support cryptographically secured API or have design incompatibilities with financial internet's accounting system should be bridged to represent their differences

The difference between traditional architecture and the new one is that the traditional approach relies on a belief that administrators (or modules that have admin rights) are following the rules. The new architecture allows processing and auditing the financial data that is mathematically verifiable and doesn't require trust towards an individual

In general, the integration with external systems is one of the most sought after features on an asset management platform. To implement these functions, the system must have an external systems integration module (or a set of modules). The purpose of this integration module is to mediate between the platform and any external financial systems.

When depositing or withdrawing fiat currencies, external systems are usually the client's bank, certain payment gateways (e.g., PayPal, SafeCharge, etc.) or other financial institutions. In this case, an integration module communicates with an external system and initiates the tokenized asset issuance (or redemption) in accordance with the data received from it.

In the case of digital currency or cryptocurrency deposit/withdrawal, both the digital currency system (e.g., Ripple, Stellar, etc.) and cryptocurrency (Bitcoin, Ethereum, etc.) act as external systems. In this case, an integration module monitors the change in the state of an external accounting system and, based on the data obtained, initiates the issuance (or redemption) of tokenized assets on a tokenization platform.

This section describes the operating principles of integration modules. It focuses primarily on the security of the mechanisms as well as on the methods employed to separate the module into several components to independently perform separate functions.

Integration modules primarily consist of four mechanisms. The first two mechanisms describe the user interaction, payment service, and asset management platform that uses the PSIM (payment service integration module) to perform deposit and withdrawal of fiat currency. Herein, neither the platform nor the PSIM processes (or stores) the users‘ private data (bank card data and other sensitive details). Therefore, the security and responsibility of managing sensitive and personal user data rely on both the payment service and user.

The other two mechanisms describe the interaction of the platform with external digital currency and cryptocurrency systems. Most major approaches define the division of a module into separate components, each performing different functions. During their interaction, the components achieve mutual consent through the use of multisignature.

Depositing fiat currencies to the platform

To deposit funds on the platform, a set of requirements have to be met. Firstly, the asset issuer must have an account in the external system. This account will be used to anchor the tokenized assets that will later be issued on the platform. Secondly, the corresponding asset should be created on the platform, and the corresponding restrictions should be configured. Third, an administrator account (or a group of administrator accounts) with the permission to issue tokenized assets has to be created and must confirm the issuance of the corresponding asset.

As we noted earlier, during a deposit, the PSIM communicates with the external financial system and initiates the issuance of a corresponding amount of asset to the user's balance. Figure 10.9 shows the deposit workflow.

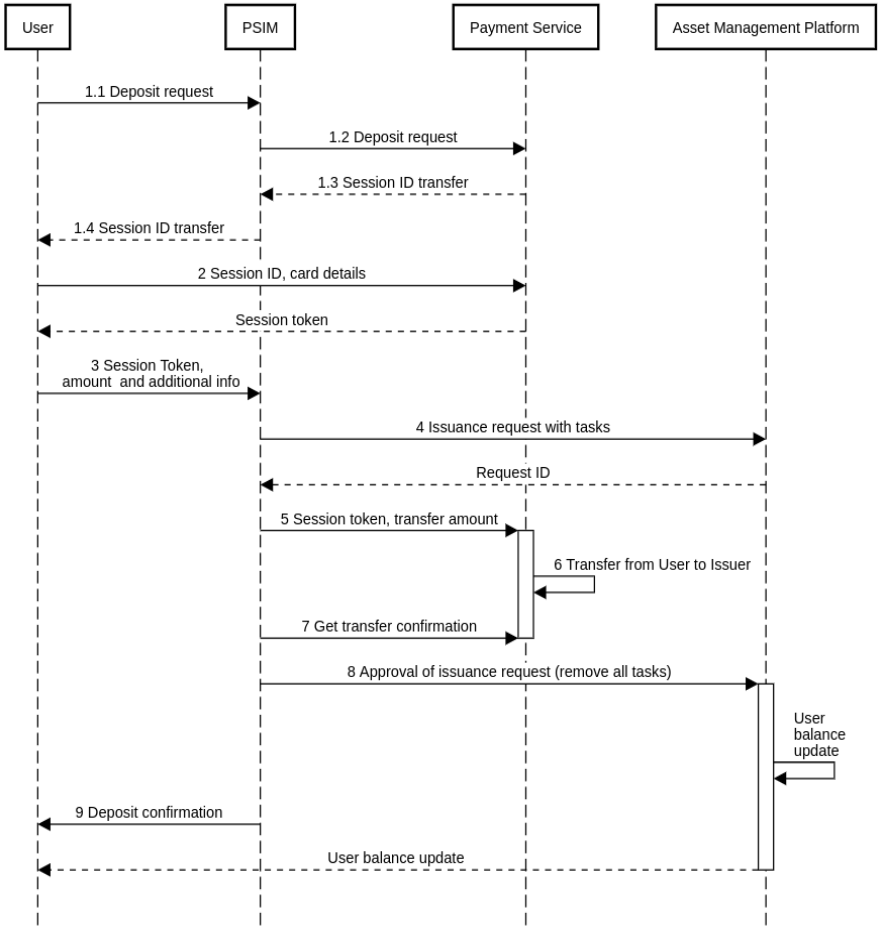

Figure 10.9 — Fiat currencies deposit workflow

The fiat deposit process consists of the following steps:

1. User receives the *session ID*.
   1.1. User sends a deposit request to PSIM. The request indicates the asset User desires to deposit.

1.2. PSIM checks the possibility of depositing the asset and sends a deposit request to Payment Service (external system).
1.3. Payment Service generates a unique *session ID* and sends it to PSIM.
1.4. PSIM passes the received identifier to User.
2. User communicates with Payment Service, sends the *session ID*, which was received earlier, and reports the details related to their account to the external system (card number, account, etc.). Payment Service processes the card details and sends a new unique identifier—*session token*, which is associated with the user's external account details—to User.
3. User sends *session token* she received and the amount of assets to be issued and transferred to PSIM.
4. PSIM sends a request to issue tokenized assets to AMP (Asset Management Platform). The request contains a set of tasks requiring processing prior to asset issuance and transfer. AMP receives the request, generates a unique request identifier and returns it to PSIM.
5. PSIM sends information about the amount of transfer and *session token* (which is associated with the user account in the external system) to Payment Service.
6. Payment Service processes the request and makes a transfer from the user account to the external account of the asset issuer.
7. PSIM receives a transfer confirmation from Payment Service.
8. PSIM confirms the issuance request on Asset Management Platform. User's balance on the platform is updated in accordance with the details of the confirmed request.
9. PSIM informs User that the deposit has been confirmed. For verification purposes, User can log in to the platform and retrieve information on his balance.

Fiat currencies withdrawal

Similarly, to withdraw funds from the platform, a set of requirements must be met. First, the user has to have an active account in the external system (this account will be used to receive withdrawals). Moreover, the corresponding asset

must be available for withdrawal. Again, there is a distinct transaction workflow involved as shown in Figure 10.10.

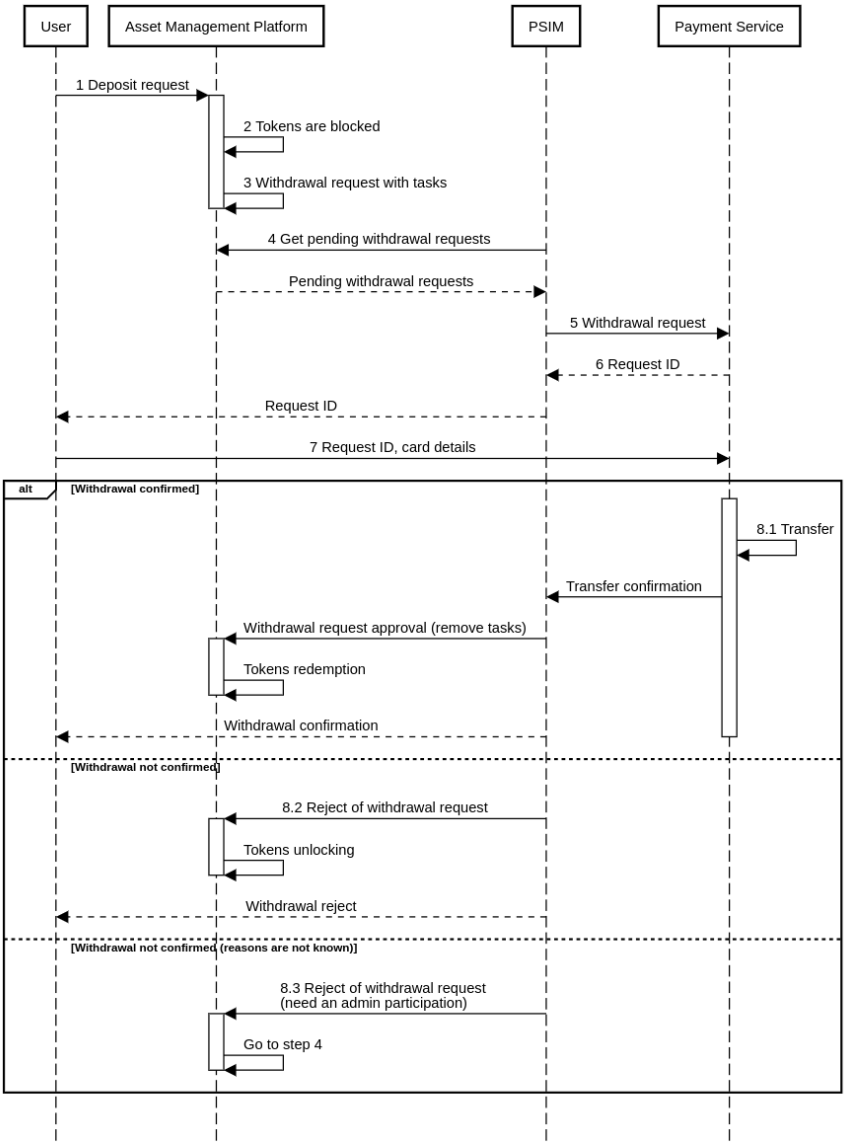

Figure 10.10 — Fiat currencies withdrawal

The fiat withdrawal process consists of the following steps:

1. User sends a withdrawal request to Asset Management Platform. The request must contain information about User's account, the amount to withdraw, and the asset to withdraw, etc.
2. The corresponding amount of tokenized assets is locked up on Asset Management Platform.
3. AMP creates a withdrawal request, which may contain certain tasks that must be performed by the platform administrators.
4. PSIM monitors such events on the platform and receives information about unprocessed requests.
5. PSIM sends information about the withdrawal request to Payment Service (amount of transfer, issuer account).
6. Payment System generates unique *request ID* value and transfers it to PSIM. PSIM transfers the received value to User.
7. User sends *request ID* and account data (card number, details, etc.) to Payment Service.
8. The payment system is processing the request. In this case, there are three possible scenarios.
   - 8.1. Withdrawal is confirmed. If Payment Service confirms the withdrawal, it transfers the requested funds to the client and sends the withdrawal confirmation to PSIM, which, in turn, confirms the withdrawal on the platform; as a result, the assets which were locked up on User's balance are redeemed.
   - 8.2. Withdrawal is not confirmed due to known reasons. If Payment Service refuses to confirm the request (e.g., it is technically impossible due to certain reasons or the user has specified incorrect details), then PSIM appeals to the system to perform the reject withdrawal operation. As a result, assets which were blocked on the user balance are again available for the user.
   - 8.3. Withdrawal is not confirmed, and the reasons are unknown. If the withdrawal request has not been confirmed by the payment service, and PSIM hasn't received any details why the withdrawal

was rejected, then PSIM requires a permissioned admin to take a decision regarding possible next steps.

Deposit of digital currencies and cryptocurrencies

Deposit and withdrawal of digital currencies and cryptocurrencies are also implemented by using the PSIM. However, in this case, the module should be divided into four components: Deposit PSIM, Deposit Verifier PSIM, Withdrawal PSIM, Withdrawal Verifier PSIM. The functioning of the first two will be described in this subsection, the description of the latter in the following subsection.

To make a deposit of cryptocurrencies or digital currencies on the platform, a set of requirements should be provided. Initially, the system administrator should create the corresponding asset and create a pool of external system account ids into the config of the Asset Management Platform core. Then, a sufficient amount of asset should be available for issuance, and the user's account permissions must allow the receipt of the requested asset.

As noted earlier, during a deposit, the PSIM communicates with the external system and initiates the issuance of the corresponding amount of tokenized assets to the user's balance. Communication takes place as shown in Figure 10.11.

The deposit process consists of the following steps:

1. User sends a request to Asset Management Platform to receive the payment account/address in External System.
2. User transfers the needed amount of funds to the received account/address.
3. Deposit PSIM keeps track of incoming transfers and their senders' accounts/addresses.
4. Deposit PSIM requests Asset Management Platform to obtain User's account linked to an external account from which the deposit has been made.
5. After receiving User's account data, Deposit PSIM creates an issuance request and sends it to Asset Management Platform core. In addition to an asset and the amount of asset to issue, the request also specifies

tasks that are required to be processed before the actual issuance takes place.

6. Deposit Verifier PSIM monitors unconfirmed issuance requests on AMP.
7. After receiving the request, the Deposit Verifier PSIM monitors External System and waits for the required number of transaction confirmations.
8. Once the required number of confirmations is reached, the Deposit Verifier PSIM confirms the issuance of tokenized assets to the user's balance, and User receives information that the deposit is successful.

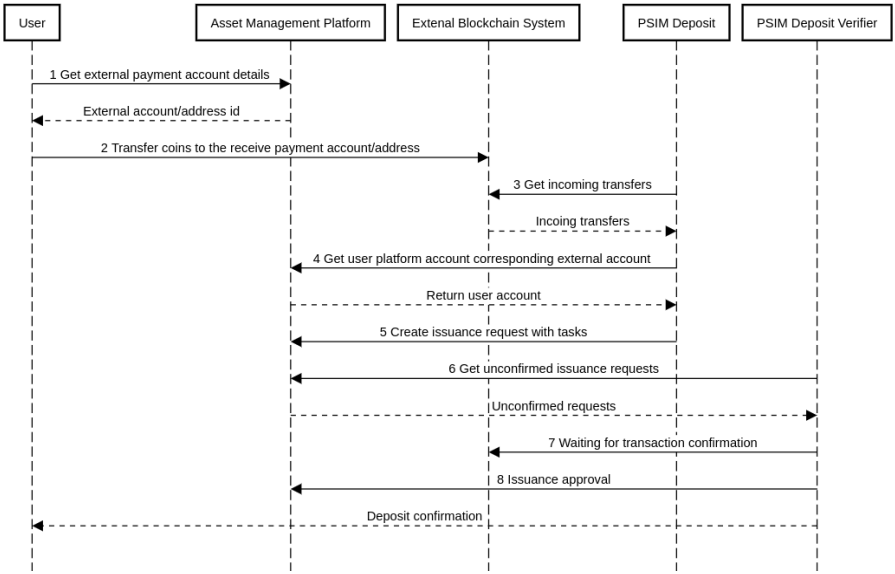

Figure 10.11 — Deposit of digital currencies and cryptocurrencies

Digital currencies and cryptocurrencies withdrawal

Withdrawal is the process of retrieving funds from the user's balance; it implies the redemption of tokenized assets on the asset management platform which is followed by a user receiving the corresponding amount of funds in an external blockchain system. The withdrawal process is shown in Figure 10.12.

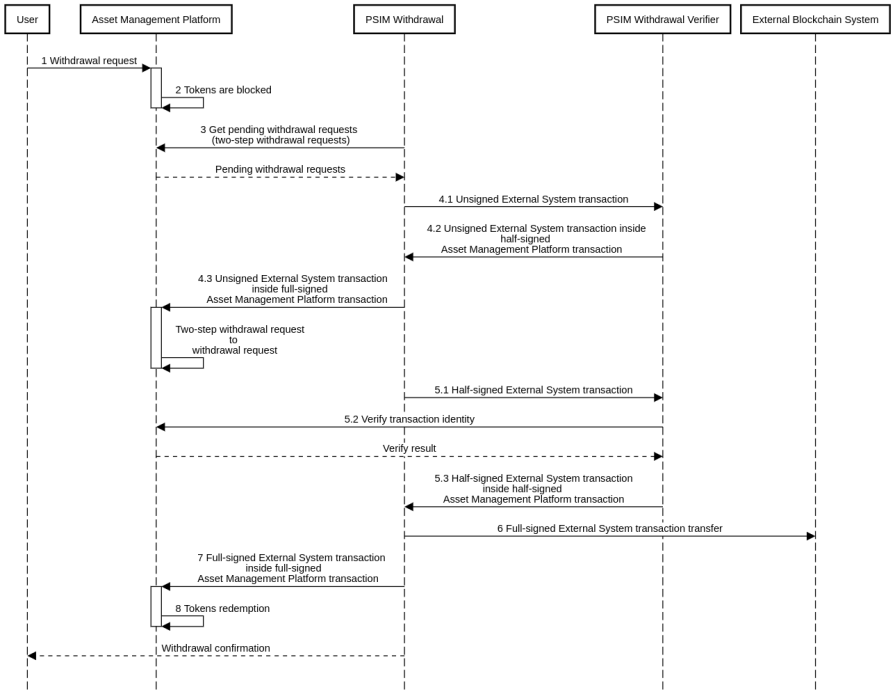

Figure 10.12 — Withdrawal of digital currencies/cryptocurrencies

The withdrawal process consists of the following steps:

1. User sends a request to Asset Management Platform. The withdrawal request contains an identifier of the balance from which funds should be withdrawn, amount to withdraw, asset to withdraw, etc.
2. The corresponding amount of funds is locked on User's balance.
3. Withdrawal PSIM notices the pending withdrawal request and starts processing it. Note that in order to confirm the request, it is necessary to perform two iterations (in this case, multisignature is used for sending both Asset Management Platform's and External System's transactions).
4. Withdrawal PSIM communication with Withdrawal verifier PSIM
   4.1. Withdrawal PSIM sends an unsigned External System's transaction to Withdrawal Verifier PSIM component.

4.2. Withdrawal Verifier PSIM receives an External System's transaction and places it to AMP's transaction. Then it signs the AMP's transaction and sends it back to Withdrawal PSIM. Recall that for AMP's transaction to be assured, it must be signed with a second key.
4.3. Withdrawal PSIM receives the half-signed AMP's transaction, verifies it, and signs it with its key. After that, it sends a signed AMP's transaction (with an unsigned External System's transaction inside) to the platform. After that, the first iteration is completed (Two-step Withdrawal Request is changed to the Withdrawal Request).
4.4. Withdrawal PSIM signs an External System's transaction and sends it to Withdrawal Verifier PSIM.
4.5. Withdrawal Verifier PSIM receives a half-signed External System's transaction and sends a request to the platform to verify the identity of the received transaction with the one that was placed in the first AMP's transaction.
4.6. If the transactions match, Withdrawal Verifier PSIM signs it with its key and puts it into new AMP's transaction. It also signs this AMP's transaction and sends it to Withdrawal PSIM.

5. Withdrawal PSIM receives a half-signed AMP's transaction with a fully signed External System's transaction in it. After that, it sends External System's transaction to External System for confirmation.
6. Withdrawal PSIM signs AMP's transaction (making it fully signed) and transfers it to AMP, thus completing the withdrawal process.
7. Asset Management Platform receives a signed transaction and redeems the locked funds.

10.4 Integration of KYC / AML providers

The main principle of KYC / AML checks is that they should not be embedded in the logic of transaction processing but rather act as external data providers that authenticate their every decision through co-signing the transactions

Anti-Money Laundering rules

One of the primary requirements for accounting systems that operate with regulated assets is to comply with the AML rules. The purpose of AML is to simplify the tracking of all suspicious actions related to money laundering, market manipulation, and other violations (e.g., the financing of terrorism).

Therefore, a regulated system must be designed in a way to perform reliable user identification as well as to trace and report all actions related to a rules violation. In this book, we will not consider how the system should be regulated. Instead, we will showcase architecturally how and which transaction verifications mechanisms should be implemented in the system to make it scalable, secure, and transparent.

Know Your Customer procedure

Know Your Customer (KYC) is a client identity verification procedure. Its end goal is to determine users' permissions for specific operations in the system. Essentially, the KYC procedure identifies and verifies a subject's identity based on the information received. It presumes three key entities:

- *Data required to conduct the procedure*
- *Method of verifying the received data*
- *Access level granted to the identification object to perform system operations*

These are closely related to each other, and they are all defined by the service provider. Hence, depending on the service provider and each specific case, the KYC process may vary. For this reason, an identification and certification platform shouldn't necessarily provide some universal and ready-made solution, but rather come with the required toolset that allows both the implementation of a proprietary solution and the integration with third-party services (delegating the KYC process to external, trusted organizations).

The process of integration with external KYC services is presented in Figure 10.13.

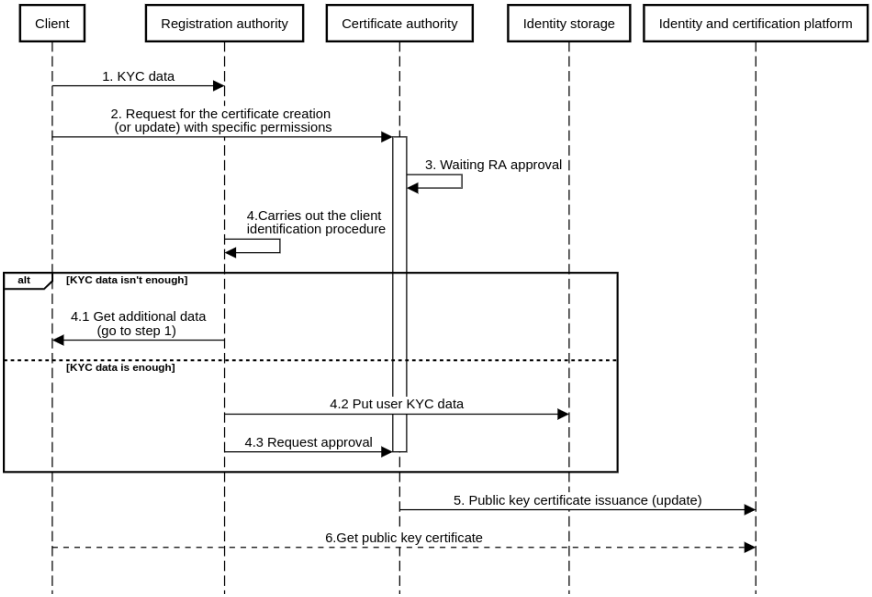

Figure 10.13 — KYC procedure workflow

The process of passing the KYC is as follows:

1. User sends a set of personal data to Identity Provider. To prevent third parties from accessing the data, it is sent to Registration Module encrypted.
2. User creates a request to get (or renew) a public key certificate or an account (depending on the platform). The request indicates the purpose of creating (renewing) the certificate and the permissions that the user wants to get.
3. Identity Provider waits for the request confirmation from Registration Module.
4. Registration Module verifies the client's identity based on the data obtained and the requested permissions.
   4.1. If the data provided is insufficient, Registration Module rejects the request (and indicates the rejection reasons). It also informs Identity Provider about the refusal.
   4.2. If the data provided is sufficient, Registration Module encrypts it

and places it in the identity storage.

4.3. Registration Module informs Identity Provider that User is authenticated and that a specific certificate can be created (renewed).

5. Identity Provider issues a public key certificate to the platform and then waits for other validators to confirm the certificate and reach consensus regarding the ledger state.
6. User can verify that her certificate has been added to the chain of blocks. From this moment, the public key of the certificate can be used according to the permissions granted.

Identity storage

Identity storage is a GDPR-compliant module which stores data collected during the KYC procedure. Identification data is stored in an encrypted form and only the registration module (or other parties who have previously received permission signed by the client) can process its personal data.

10.5 Reconciliation of assets between issuer banks

Until now reconciliation often occurred manually or semi-manually in the case of errors. This slowed down business processes that required internet speed

Let's consider an example. Alice who is a client of Bank 1 wants to send 100 USD to Bob who is a client of Bank 2. If these banks both use a shared platform for maintaining the digital currency infrastructure where they account for, say, the US-dollar-backed asset (further referred to as T-USD), then it is no longer a problem for Alice to send money to Bob directly and with minimal fees (Figure 10.14).

NOTE

This tokenized dollar or, as referred, US-dollar-backed asset is, in fact, the very same dollar and only the way it is accounted for has changed—this is due to the same reason we do not draw distinctions between paper dollars and digital dollars (it always remains dollar).

However, in the below flow, we will refer to it as *T-USD* exceptionally to avoid the confusion and to make it clear when we refer to whatever is accounted on the Asset Management Platform. In real life, this will be called dollars obviously.

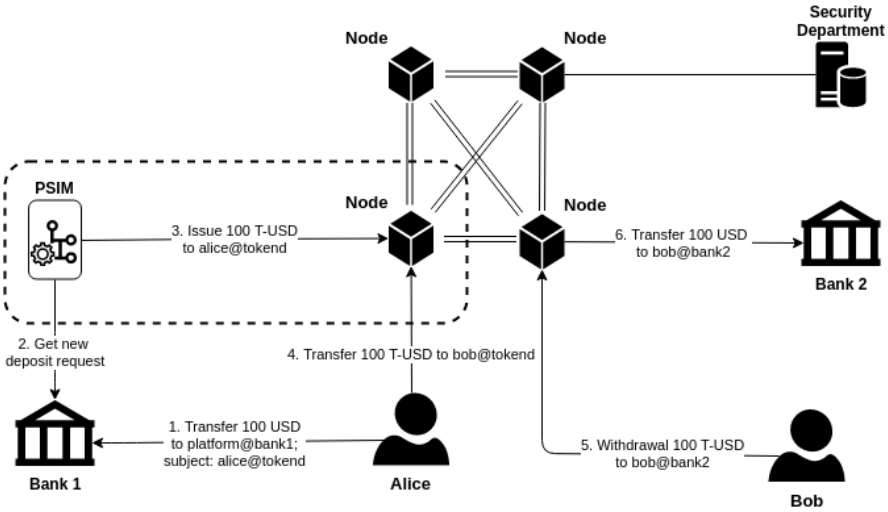

Figure 10.14 — Reconciliation between partner banks

The process of reconciliation entails the following steps:

1. Alice performs a deposit of 100 USD to her Bank.
2. Platform's payment system integration module receives a new deposit request that includes Alice's account identifier and the deposit amount.
3. Payment system integration module initiates the issuance of 100 T-USD to Alice's wallet on the platform.
4. Alice transfers 100 T-USD to Bob's account on the platform. Most notably, compared to traditional systems, the "magic" happens at this very moment: the distributed ledger technology acts as a trustless bridge between the two payment systems, meaning that this transaction of Alice is recognized in real-time and verified by both banks. The banks can easily update their own database entries

correspondingly, with Alice having 100 USD less on her Bank 1 account and Bob having 100 USD more on his Bank 2 account. Noteworthy, the state of correspondent accounts between Bank 1 and Bank 2 is updated based NOT on the information from, for example, a central bank or SWIFT but based on the real-time consensus between the banks.

5. Bob makes a withdrawal request for 100 T-USD, and 100 USDs are transferred to Bob's bank account.
6. Meanwhile, the security department while not being able to interfere with the entire process, has the authorization to perform an explicit audit of the performed transaction and to make sure that the actions performed within the platform adhere to all the regulatory requirements.
7. At the end of a certain period, Bank 1 initiates a wire transfer of 100 USD (or whatever is the netting result of user transactions between Bank 1 and Bank 2 during this period) to Bank 2. The platform's shared ledger is considered to be the primary source of information on the state of the correspondent accounts.

11 PRIVACY

11.1 General Data Protection Regulation

GDPR states an ambitious and right goal: only the users have to be the sole owners of their personal data

Nowadays, most companies are legally required to store and process the personal data of their users. While the users' personal data is of great value, some organizations have a practice of exploiting these data improperly.

Given some companies' exploitation of their users' data, it is crucial that the confidentiality of the data is ensured. For this purpose, the *General Data Protection Regulation* (GDPR) was ratified—a set of rules that establish how users' personal data must be processed and stored when collected [14]. All organizations that provide services in the European Union must comply with the GDPR. In this section, we consider the main guidelines of the GDPR, and in the following, we will look closer at the details of technical solutions.

What is the users' personal data? In fact, the personal data of some subject is a set of information that is directly or indirectly related to the subject and can be used to identify it. Some examples of personal data are name, surname, and biometric data, while the indirect data are an email address, IP-address, etc. (Figure 11.1).

The regulation describes the following conditions:

- *Rights of end users whose data is being processed*
- *Features of storing and processing personal data*
- *Features of sharing personal data with third parties*
- *External auditors and regulators features of work as well as of their staff*
- *Dispute resolution mechanism*
- *Types of violations regarding personal data and responsibility issues*

One of the most important requirements regarding compliance with the GDPR for organizations is that there must be a justification for storing and processing users' personal data. That means an organization must prove that

the data which it processes is actually needed to provide services to its clients (while the set of the data processed must be strictly limited).

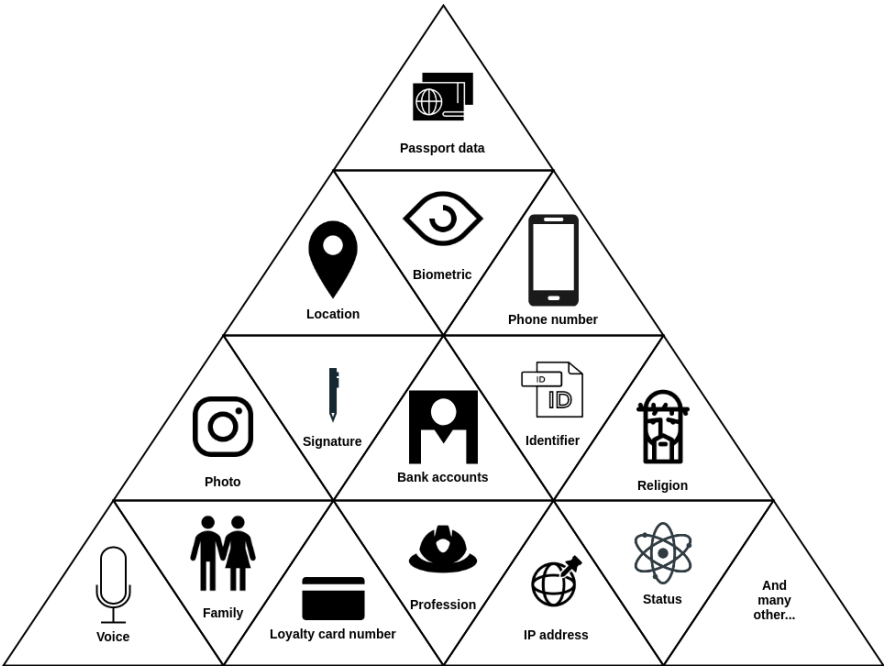

Figure 11.1 — User personal data

Another key requirement is that users must be provided with full control over their data. This includes the following [50]:

- *Providing up-to-date information about the data stored (including storage period, conditions and processing details)*
- *Deleting all personal data upon the user's request*
- *Correcting any personal data upon the user's request (if they are not accurate)*
- *Terminating the processing of the data upon the user's request*
- *Providing a full copy of the data on the user's request*
- *Determining by a user how her personal data can be processed*
- *Sharing personal data with other parties upon the user's request*

Also, much attention should be paid to the reliability of the CISS built to preserve integrity and confidentiality of users' personal data and to the monitoring of the violations (as well as violation attempts) of the listed security services. The measures to protect this data must be applied as early as possible during the stage of building a system / platform / service and not after the creation of a ready-made product.

All actions for processing personal data must be documented. The reasons and methods for storing and processing personal data must be documented, and audit logs revealing individuals who accessed the personal information must be maintained. Documenting all security breaches related to the integrity and / or confidentiality of users' personal data is also of great importance.

11.2 Limiting access to confidential data

Proving certain aspect of your personal data must not reveal all the aspects

In this section, we describe the solution to a very practical problem of identity verification, which has a well-known analogy in the physical world: when the bartender wants to verify your age, you don't want to disclose your address and name, so just your picture and year of birth is enough.

As noted earlier, organizations that collect the user's personal data are responsible for the security of this data storage and processing. The issue of transmitting this data to third parties is particularly critical. According to the GDPR, you can only share the personal data which a user to whom this data belongs has given permission to share. And while it is possible for a third-party to verify the integrity of the full set of data (through matching hashes), the task of verifying the integrity of only the part of data, which the third-party has received permission to receive, is much more complex. In this section, we will describe methods to solve these issues.

Creation of permission to receive personal data

In order to obtain a user's personal data from the registration module that performed the initial identification of this user, the requesting organization must first receive the user's permission. The user creates a request and sends it to the

organization requesting her data (so that this organization could later send it to the registration module). Let's consider the details that must be specified in the permission (Figure 11.2):

1. Recipient's identifier. This field can contain the recipient's public key, which encrypts the personal data before it is shared. Thus, no other party will be able to access it.
2. What kind of personal data may be shared. During identification, a user can provide any specific data, which should not be shared with unwanted parties. Therefore, in the permission, a user must specify which of the data can and cannot be shared.
3. User's signature. By providing the digital signature generated with her private key, the user confirms the permission to access her personal data (the signature guarantees that permission details cannot be modified)

Figure 11.2 — Details of a user's permission

The permission may additionally contain some other metadata, but it is not that important compared to those listed above.

Using the Merkle tree to authenticate users' personal data

As mentioned in section 7, in order to obtain an account, a user must be identified by the registration module. The user provides the registration module

with a set of his personal data. The hash of this data will be included in the account. This value can be used in a way that the user himself or another registration module or even a separate application server can verify that the user's personal data was processed correctly and has not been modified during processing.

To obtain personal data, a third party must obtain the necessary permission from the user. However, very often the requested party needs much less data than it was provided, and a user does not want to give access to all his personal data. For example, a user was registered via the medical institution's registration module, and in addition to the first name, last name and passport data, the identity storage also stores the user's medical data, while the bank's registration module additionally wants to verify the correctness of the passport data.

One of the proposals for solving this problem is using the Merkle tree structure. In this case, during the identification, all personal data is split into blocks, which are further added up to the structure of the Merkle tree (Figure 11.3). The Merkle root value (root node) is placed in the certificate field, *hash of personal user data*.

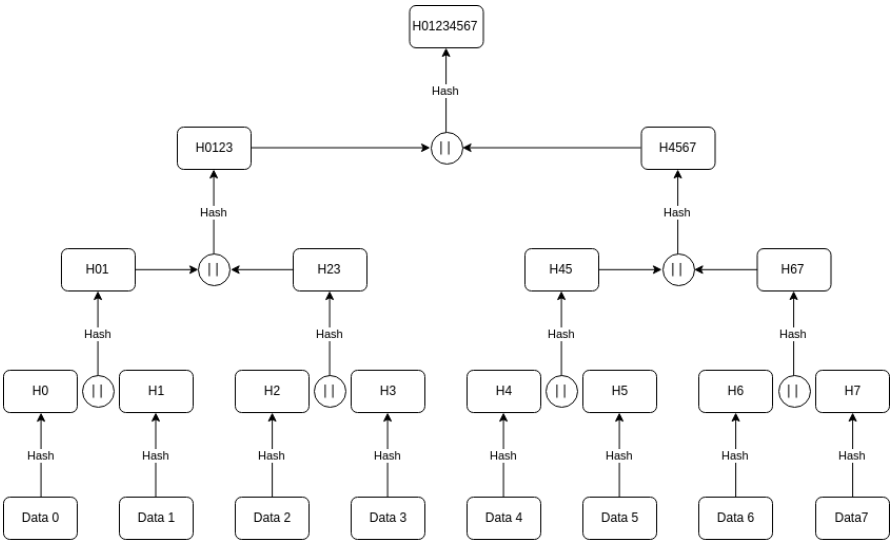

Figure 11.3 — Merkle tree structure

If the verifying party requests only one piece of data, the registration module returns the desired piece of data and the Merkle branch value for this piece of data to it (Figure 11.4). Having the data piece and the Merkle branch value, a verifier can independently calculate the Merkle root value to verify that it matches the one recorded in the public key certificate.

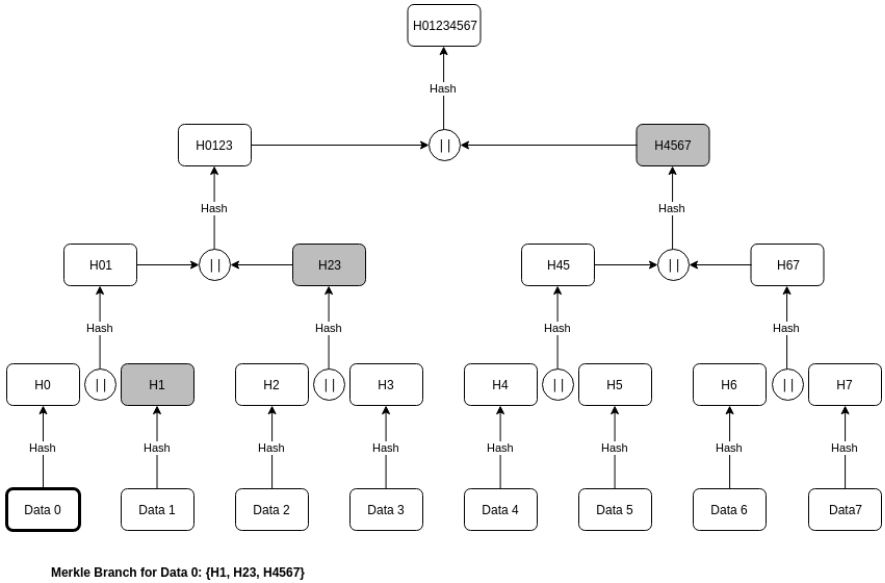

Figure 11.4 — Data authenticity verification path

However, there may be a situation when a verifier requires several pieces of the user's personal data. In such a case, if the registration module sends Merkle branch values for each data fragment, some values of the Merkle tree nodes may overlap, resulting in redundancy. To solve this problem, multiple authentication scheme in a Merkle tree called Octopus Authentication [51] can be applied. The scheme allows creating a Merkle branch while avoiding redundancy, namely through increasing the number of authentication nodes (Figure 11.5).

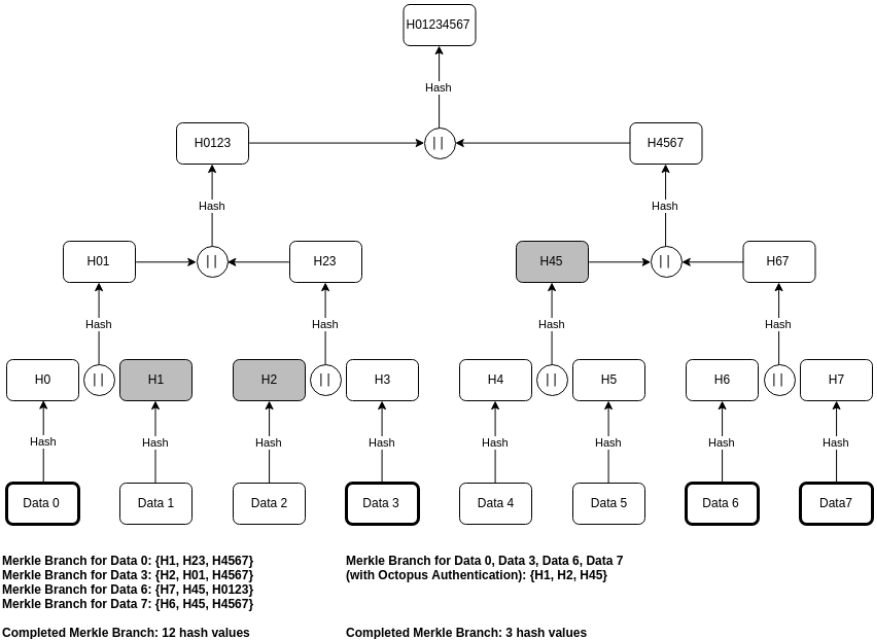

Figure 11.5 — Data authenticity verification with Octopus Authentication

11.3 User privacy

For engineers with a financial background, here we describe how users can securely interact without initiating transactions in the accounting systems of their custodians

For blockchain engineers, here we describe the simplified version of payment channels / Lightning Network that can be implemented in an environment where every system has real-world identity and reputation

By default, all transactions (and operations) on the asset management platform are visible for all platform validators (and auditors). This feature simplifies the verification of transactions history and asset management in general (verification of user permissions for each operation). However, this approach negatively affects the confidentiality of payments, which is very often a critical requirement for businesses and their customers.

What would be the best option then to resolve this conflict? The first and most obvious solution is to build the system in a way that only validators and auditors have access to the database of all transactions and to the final state of accounts. Such a solution can be considered optimal in terms of the accounting system functionality, performance, and responsibility distribution. However, this solution presumes much lower transparency and also introduces the risk that validators will disclose transaction details.

Payment channel approach

Another suggestion for increasing the privacy of transfers between platform users is to use a payment channel mechanism. It is more difficult to implement and much more demanding in terms of client applications (digital wallets). However, this approach can significantly increase the privacy of transfer and exchange operations by moving the business logic of some transactions outside the platform. A payment channel does not require sending transactions to validator nodes as well as disclosing transaction details and paying fees. Only the results of parties' interaction are published on the chain of blocks, while all the intermediate stages are known only to the counterparties.

How can the support of payment channels ensure the confidentiality of payments and exchanges while meeting the regulatory requirements? Let's examine how the approach works and explain how its security is achieved and how it is ensured that the channel participants cannot commit fraud.

Imagine two parties, Alice and Bob, who open a payment channel between them (Figure 11.6). Initially, Alice and Bob both have two types of tokenized assets on their balances, T1 and T2.

1. Alice creates transaction 1, which locks 100 coins of each type on the users' balances and sets conditions that the locked funds can be distributed only if both parties' signatures are present in the transaction (if the unlocking transaction contains *endflag* set to TRUE, then the transaction is confirmed immediately, and if FALSE, then it is confirmed in 24 hours). The transaction is signed with Alice's key.
2. Alice creates transaction 2, which spends the locked coins (distributes them back to balances of Alice and Bob). This is done to allow parties

to unlock the funds because otherwise they would have remained locked forever. The transaction also contains the hash value of the locking transaction (transaction 1) as well as the hash of the previous distribution transaction (in such a case, this value is zero since this transaction is the first one which is unlocked).

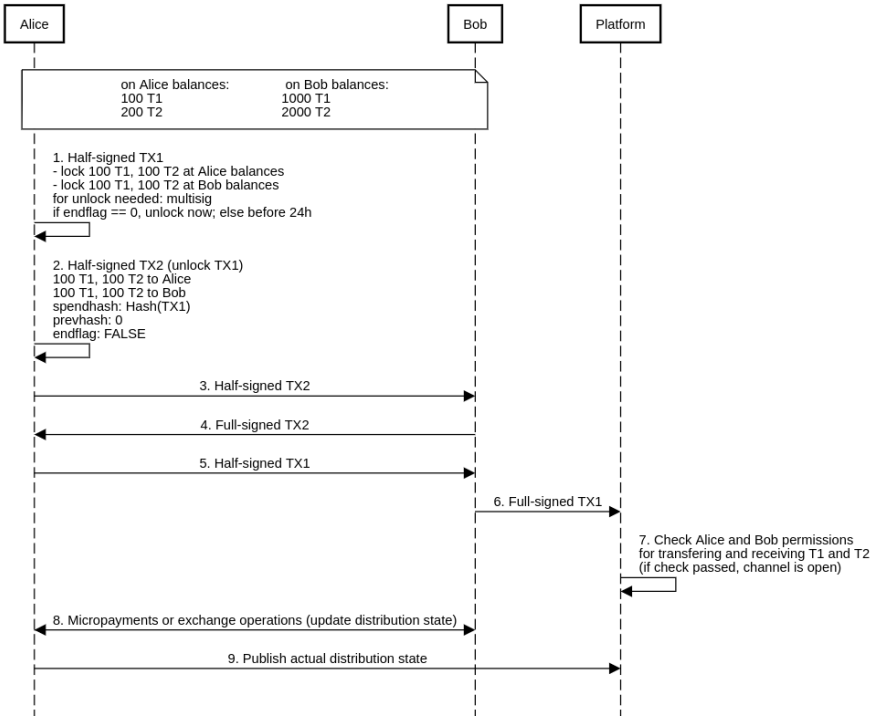

Figure 11.6 — Payment channel workflow

3. Alice sends transaction 2 to Bob.
4. Bob verifies the transaction, agrees with the initial distribution and returns the fully signed transaction to Alice. Now, this transaction can be published by any of the parties to return funds to the balances (returning balances to their original state).

5. Alice sends transaction 1 (the locking one) to Bob. Bob verifies the transaction and adds his signature to it.
6. Bob publishes a fully signed transaction 1 on the network.
7. Validators verify the transaction. Another fact verified is that both Alice and Bob have permissions to send and receive the funds specified in the transaction. If they both have such permissions, validators confirm the transaction and lock the coins. Now neither Alice nor Bob can spend them in other transactions.
8. Alice and Bob conduct micropayments and exchange operations between each other thus constantly updating the state of an unlocking transaction. Every updated transaction state must contain the previous state hash (so that neither of the parties can send the old distribution state to the network and confirm it) and must also be signed by the two parties. Therefore, payments are conducted in very small amounts (micropayments), to prevent any interacting party from stealing the vast amount of coins at a time. When Alice and Bob decide to close the channel, they change the flag of the last transaction, sign it, and send it to the network.
9. The transaction is confirmed. Coins are distributed between the user accounts based on the last transaction data.

Why can't one party cheat another? If one of the parties publishes an outdated transaction, then since its flag is set to FALSE, it cannot be confirmed earlier than in 24 hours after validators receive it. When the second party detects the fraud first, it can send an up-to-date transaction or a chain of transactions that leads to the one sent earlier (each transaction contains the hash value of the previous state) to validators. In this case, confidentiality could be partially violated, but, nevertheless, neither of the parties can steal a meaningful amount of coins from the other. The exchange transaction between two parties completely excludes the possibility of fraud since parties agree to change the state with only one transaction.

It is possible to build a network on top of a platform based on such channels (something similar to Lightning Network [52]). The transfer of coins from one party to another will be performed by changing the states of all the

intermediaries' channels. Let's imagine that there is one open channel between Alice and Bob and another one between Bob and Carol (Figure 11.7).

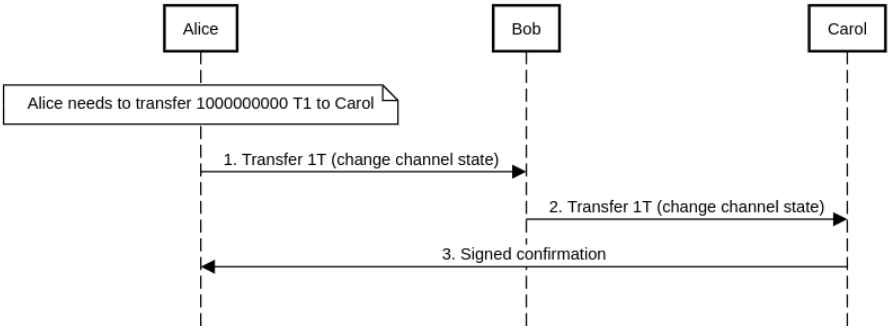

Figure 11.7 — One of the methods of channels commutation

1. Alice needs to transfer some coins to Carol. She has an open channel with Bob, and she knows that Bob is connected to Carol. Therefore, Alice sends him 1 coin via this channel (a transaction changes the channel state).
2. Bob receives Alice's coin and sends it to Carol (note that this is the same coin, but it is now sent through the channel between Bob and Carol).
3. Carol receives the coin and sends an incrementing signed transfer confirmation to Alice. Alice receives the confirmation and sends the next coin. Note that fees are zero (even though Bob could set minor fees for him acting in the role of a bridge).

Even though Bob could indeed technically outwit Alice, he can only succeed in stealing 1 coin (its value is much less than the total fee amount and worth less than the potential loss of reputation).

Similarly, cross-exchanges can be conducted (Figure 11.8).

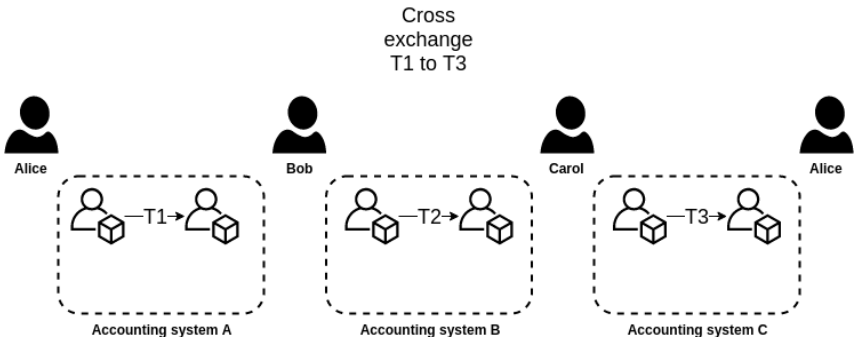

Figure 11.8 — Cross-exchange scheme

Let's observe how this works (Figure 11.9). Initially, Alice agrees with Carol that they want to exchange 3 T1 for 2 T2. Bob is an intermediary between them.

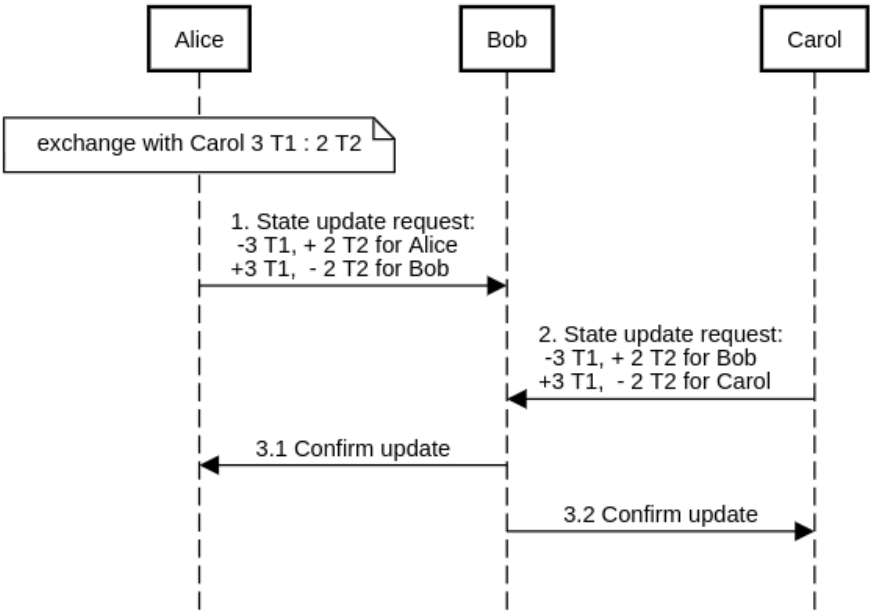

Figure 11.9 — Cross-exchange workflow

1. Alice sends a partially signed transaction with an exchange offer to Bob.

2. Carol also sends a partially signed transaction with an appropriate exchange offer to Bob.
3. Bob verifies that the transactions match and confirms them (the total number of Bob's coins has not changed while amounts in the channels have changed).

If Bob does not confirm both transactions, then the exchange won't be performed. If Bob signs only one transaction, then it means that only one of the parties has made the deal (but haven't lost the funds).

What are the benefits of such channels?

- Parties act completely privately (transactions can be exchanged in an encrypted form)
- Interaction results are published in the ledger
- Regulation is not limited (user permissions are verifiable, and limits can be adjusted)
- Parties cannot cheat (they can, technically, but this is meaningless since the outcome is not worth it). And in the case of an exchange, even the possibility of fraud is absent.

11.4 Anonymity in digital currencies

Transaction metadata (even the fact of its existence) can be private but can be proven by any involved party

In this section, we consider basic mechanisms for ensuring the confidentiality of transactions and user privacy in accounting systems. Their characteristics are that they mostly use the accounting model based on the one-time addresses and the unspent transaction outputs (UTXOs). Such mechanisms cannot be applied in pure form for the asset management platforms which use account-based and balance-based accounting models. However, we believe it is important to consider the basic concepts as they are crucial in building the tools that guarantee the privacy of the user on asset management platforms.

Data that require confidentiality

- *Transaction parties' identifiers*
- *Transfer amount and the asset identifier*
- *Additional data*
- *Transaction conditions (smart contract content)*

Blind signatures

The blind signatures mechanism allows users to get permission to perform specific operations without disclosing their details [53].

This mechanism can be used for the bank transfers between users, and the bank will not have the details of the recipient (and in the case of withdrawal, coins sender details).

The blind signatures mechanism works as follows (Figure 11.10):

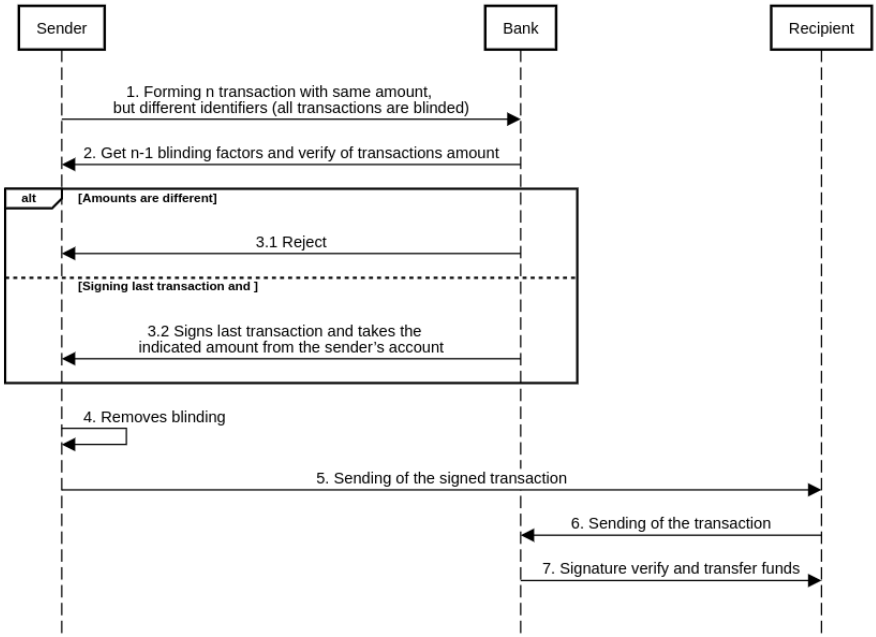

Figure 11.10 — Blind signature workflow

1. Sender creates a set of transactions with different identifiers but with the same transfer amounts. Each transaction is blinded (encrypted) with a separate key. After that, he passes the entire set to Bank.
2. Bank requests ($n-1$) random keys used for blinding. Sender transfer the required keys.
3. Bank decrypts the set of transactions according to the received keys and verifies that the transaction amounts are the same and that Sender has enough funds on his balance.
   3.1. If amounts in transactions differ (or the sender does not have enough funds), the request is rejected.
   3.2. If amounts are the same, Bank signs the last transaction while not disclosing its content and sends it to the sender. After that, Bank withdraws the specified amount from Sender's account.
4. Sender removes the blinding from the transaction; at that, the blinding from the signature is also removed (now the signature is valid in relation to the initial transaction).
5. Sender broadcasts the transaction signed by the bank to Recipient.
6. Recipient applies to Bank with this transaction. Bank verifies the signature and sends the funds. Note that the transaction does not anyhow indicate who the sender of funds is. Moreover, Bank will not be able to distinguish to whom this transaction was addressed since it was blinded when signed.

Stealth addresses

Stealth addresses is a method of hiding information about the recipient [54]. This method is based on using one-time keys, which are generated by the sender with random values and the recipient's address (account). To receive the funds, a participant reviews all the output keys one by one and checks which of them matches the private key she can generate (based on her master private key).

The advantage of this approach is that only the sender knows the actual address of the recipient. For everyone else the connection between the one-time key and the recipient's address is not traceable.

The stealth addresses method works as follows (Figure 11.11):

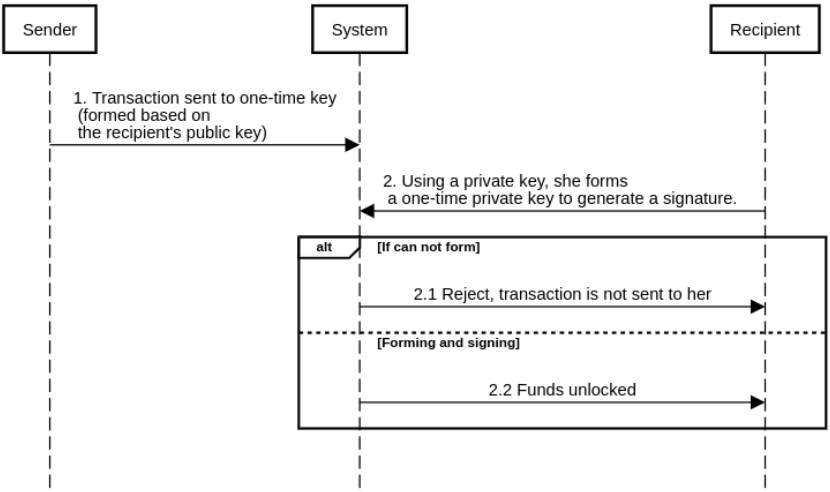

Figure 11.11 — Stealth address workflow

1. Sender creates a transaction that transfers funds to a one-time public key. One-time key is generated based on the recipient's public key. The transaction is sent to System and confirmed.
2. Recipient generates a one-time key to unlock coins by using his private key. He checks each unspent transaction output on whether he can reproduce the required signature for it.
   2.1. If the attempt fails, then the specific output was not addressed to this recipient.
   2.2. If Recipient manages to generate the correct signature for the specific output, then it is considered that she proved that she owns the funds and can transfer them.

Confidential transactions

Applying confidential transactions (CTs) allows increasing the transaction confidentiality level while still making it possible to verify the records in the system (i.e., to verify that the funds don't appear out of nowhere). CTs are built based on Pedersen commitments. These commitments allow a user to prove a certain value without disclosing it (the concept of zero-knowledge proofs)[55].

In general terms, the Pedersen commitment scheme is presented as

Hash(blinding factor || value)

or, if an elliptic curve is used,

*blinding factor*G + value*H*

where *value* is some value you know but don't want to disclose, *Hash* is the output value of the hash function, * is multiplication of a scalar and the elliptic curve base point. If the value was simply hashed (or multiplied by a base point), then a third party could have found it by iterating over it. The so-called blinding factor allows introducing some redundancy to prevent disclosing the value sent. If one needs to prove that the commitment actually matches some value, she will use her key to prove it.

Range proofs

One of the key issues here is that the calculations described above are done modulo a big number; hence, a huge amount of funds can be created out of nowhere:

44 (mod 43) = 1 (mod 43) — 43 out of nowhere

Such an error can occur only if the hidden value is greater than the modulo value. Therefore, it must be proved beforehand that the value sent is less than the module. For this, range proofs are used.

In fact, the value sent can be presented as a sequence of bits. In such a case, the necessary condition would be that the number of the hidden value bits is less than the number of the modulo bits (in fact, this can also be proved using Pedersen commitments). Thus, the hidden value must be presented in the form

$$value = value_0 + 2value_1 + 2^2 value_2 + \ldots + 2^{k-1} value_{k-1},$$

and a commitment for each $value_0$, …, $value_{k-1}$ must be created. Due to the properties of commitments, their sum will be equal to the commitment of the original value. In this way, it is proven that the number of the hidden value bits does not exceed the number of the modulo bits. What remains for the verifier to do is to prove that the value not equal to 1 or 0 has not been hidden in one of the bits. For this, the ring signature mechanism is used; the value of a ring signature proves that either 0 or 1 was used to sign each commitment while not disclosing which one.

CONCLUSION

This book wasn't born as an attempt to predict the future. It was born as a consequence, as a result, of solving the problem. The problem of the cumbersomeness of today's financial systems.

There are thousands of systems in the world; most of them are proprietary, have no specifications or APIs, and only a limited number of people know how to support them. The development of such systems more resembles craftsmanship than technology.

Throughout four years, we had been executing projects, stepping on rakes, analyzing, and improving—eventually we came up with the approaches and with our global mission.

This book is the result of this work. We defined the scope of our vision, wrote it down, and documented all the approaches applied. They might not be the best, but they work in practice today. Essentially, what we produced is a kind of a set of blueprints about how to build accounting systems in the internet age that must use cryptography.

If these approaches are extended and standardized, then this will be a great benefit for engineers who, in the course of the next 10 years, will be busy integrating similar solutions into the financial infrastructure to build secure and reliable accounting systems.

We are open to any critique and suggestions for improving our perspective. Over time, many of the concepts described can be presented in more detail because they are being gradually implemented and improved based on practical use.

GLOSSARY OF TERMS

Account

The basic structural unit on the asset management platform. Each account has a set of permissions to perform operations and a set of cryptographic keys for carrying out the listed operations.

Accounting system

The system that processes and stores transactions and uses a decision-making mechanism to update the final state of balances.

Application service

A software and/or hardware that provides data or services to other software and/or hardware devices.

Asset

Anything tangible and intangible obtaining three following properties: medium of exchange, unit of account, and store of value.

Asset management platform (AMP)

The system that accounts for the user assets. An important component of any asset management system is the decision-making mechanism for updating its final state.

Auditor

An individual who has the right to perform verification of the accounting system. In the context of the blockchain technology, an auditor downloads data from all the blocks and verifies them in order to check the final state of the accounting system for compliance with the transaction history while observing all the rules described by the protocol.

Authentication

The process of verifying the authenticity of a party by checking the authentication data provided by the party to subsequently give him access if the data is correct.

Authorization

The process of managing access of users to a particular service.

Blockchain

A secure method of storage and synchronization of a ledger between independent parties who do not necessarily trust each other.

In more detail, technology that describes a way to organize data by grouping it into blocks, each containing a hash value calculated from the data of the previous block. The chain of blocks is formed in such a way that an adversary cannot modify the previously added data without being noticed by other participants supporting it.

Certificate authority (CA)

An entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate.

Client

Part of hardware and/or software served by the server.

Container

An object that contains encrypted user keys whose purpose is to ensure the integrity of these keys.

Container cipher key

A key that is used to encrypt user keys during the creation of a secured container.

Cryptocurrency

A permissionless accounting system with all processes decentralized: asset issuance, transaction confirmation, data storage, accounting system audit, and governance (decision-making about the updates). Anyone can become a member of the system, hold assets, and perform transactions under the rules which are common for all.

Deposit

The process of depositing funds on the user's balance; it implies the issuance of assets within the Asset Management Platform with their subsequent transfer on the user's balance.

Digital currency

A digital asset that uses cryptography to secure financial transactions, control the creation of additional monetary units and verify the transfer of assets.

Fiat currency

A currency without an intrinsic value that has been established as money controlled by the government.

Identification

The process of assigning unique attributes to an object/subject; it may also indicate the process of verifying whether the provided attributes match those previously declared.

KDF-parameters

Parameters for the key derivation function required to generate the encryption key.

Key compromise

The fact of a third party exposing the user's secret key.

Key management

A set of actions related to the processing of cryptographic keys and other related parameters.

Key server

A server whose purpose is to store user private keys securely.

Keys backup

The process of creating a copy of the key data to be able to restore access to the services of an accounting system.

Ledger

A digital table that records the results of transactions execution and that reflects the final state of the accounting system obtained through processing the entire transaction history.

Master account

An account that has permission to perform all operations in the accounting system.

Operation

An instruction that changes the state of the ledger after being initiated.

Payment service

A system that offers bank and card payments services, processes paper or electronic checks, and reports on the financial operations.

Payment services integration module (PSIM)

A module (or a set of components) that plays the role of a bridge between the asset management platform and external systems such as financial organizations, cryptocurrencies, and digital currencies.

Permission

Platform-defined right that allows one to perform certain operations or have access to various modules of the system.

Public key infrastructure (PKI)

A set of roles, policies, and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage asymmetric cryptographic operations.

Registration

The procedure of a user registering on a site, server, etc., which is followed by the creation of an account in the server's database.

Registration module

An entity that provides identification service for further certification.

Role

A set of predefined permissions for each structural module of the platform. Each role contains a list of permissions that define the scope of activities that a participant who plays this role can and cannot do with the resources he uses.

Role-based access control (RBAC)

A mechanism that controls access of a system participant to a particular resource or to fulfilling operations in the system based on her role.

Safety

The ability of mechanisms to resist attempts of successful cryptanalysis, implementing threats on confidentiality, integrity, authenticity, availability, observability, etc.

Salt

A random value used to introduce entropy at the input of the hash function when receiving the user password hash value.

Secret

Some secret value required for a user to access the service. The secret can be a user-defined password, biometric data, or a secret stored by a specialized chip.

Server

Software and/or hardware that provides data to other software and/or hardware devices.

Service

A set of functions provided by a server or group of servers.

Simple payment verification (SPV)

A method that allows a client to verify that a transaction is confirmed without the need to download the entire transaction history and perform its verification.

Token

A unit of accounting that is used to represent the digital balance of some asset; the ownership of a token is proved with cryptographic mechanisms, particularly, a digital signature.

User

A party that uses network services through client software, for example, to store user keys.

User ID

Some unique value that allows a system to uniquely identify a user in the system.

User password

Character string that is used to authenticate a person to verify access authorization and to obtain cryptographic keys.

Validator

A member of the accounting system who performs the full verification of transactions and is directly involved in their confirmation. If there are multiple validators, they use consensus mechanisms to reach consensus regarding the ledger final state.

Verification

The process of verifying data for compliance with the protocol rules.

Web-of-trust

A decentralized trust model which is an alternative to the centralized trust model of a public key infrastructure (PKI), and which relies exclusively on a certificate authority (or a hierarchy of such).

Withdrawal

The process of retrieving funds from the user's balance; it implies the redemption of assets within the Asset Management Platform, which is followed by a user receiving the corresponding amount funds in the external payment system.

ACKNOWLEDGEMENTS

For the direct participation in the creation of this book, we are deeply thankful to Serhii Pomohaiev, Vladimir Dubinin, Yaroslav Panasenko, Dmytro Haidashenko, Bogdan Gryekhovodov, Mykhajlo Shaforostov, Borys Bodnar, Richard Horrocks, Christian Heinz.

Advice given by Oleksiy Varfolomiyev has been particularly helpful in correcting the text.

ABOUT THE AUTHORS

Dr. Pavel Kravchenko

Co-founder & CEO, Distributed Lab. Research interests: blockchain and decentralized technologies. Areas of expertise: cryptography, public key infrastructures, asset tokenization, security models, and decentralized systems architecture. Ph.D. in information security, author of 15 scientific papers, 5 of which are devoted to blockchain technology and decentralized systems. In 2014 Pavel started his work on creating decentralized accounting systems as a cryptographer at the Stellar project. He is a co-author of "Blockchain and Decentralized Systems", which is officially a textbook for students in technical universities of Ukraine.

Bohdan Skriabin

A leading specialist in cryptography and decentralized systems at Distributed Lab. Currently a Cyber Security Ph.D. student. Research interests: protocols of decentralized accounting systems, software wallet architectures, cryptographic schemes, and privacy in digital currencies. Since 2014 Bohdan has been involved in the industry of decentralized technologies and blockchain. Defended his Master's thesis on "Analysis of vulnerabilities of decentralized payment systems". Taught 9 academic and elective courses in the universities of technical profile. He is a co-author of "Blockchain and Decentralized Systems.

Oleksandr Kurbatov

Blockchain Researcher at Distributed Lab. Currently receives a master degree in Cyber Security. Wrote a bachelor's thesis on post-quantum digital signature algorithms. Area of interests: e-voting platforms, key management and privacy techniques in accounting systems. Author of articles about anonymous decentralized e-voting and decentralized public key infrastructures. One of the speakers in an online course on blockchain and cryptocurrencies as well as one of the organizers of decentralized technologies fan club.

USED SOURCES AND LINKS

1. Morse, E.A. and Raval, V., 2008. PCI DSS: Payment card industry data security standards in context. Computer Law & Security Review, 24(6), pp.540-554.
2. Cavage, M. and Sporny, M., 2018. Signing HTTP messages. https://tools.ietf.org/html/draft-cavage-http-signatures-11
3. Ethereum Classic is under attack. https://qz.com/1516994/ethereum-classic-got-hit-by-a-51-attack/
4. Yuhua, F., 2015. New Newton Mechanics Taking Law of Conservation of Energy as Unique Source Law. See: China Preprint service system.
5. Is a Single Global Blockchain Platform Already Coming Into the Picture? Here Are the Signs. https://hackernoon.com/is-a-single-global-blockchain-platform-already-coming-into-the-picture-here-are-the-signs-a5c906bbbaf9
6. Callas, J., Donnerhacke, L., Finney, H., Shaw, D. and Thayer, R., 2007. OpenPGP message format (No. RFC 4880).
7. Cortet, M., Rijks, T. and Nijland, S., 2016. PSD2: The digital transformation accelerator for banks. Journal of Payments Strategy & Systems, 10(1), pp.13-27.
8. Kravchenko, P., Skriabin, B., Dubinina, O., 2018. Blockchain and decentralized systems.
9. Nussbaum, A., 1957. A History of the Dollar. New York: Columbia University Press. p. 56.
10. Stellar Developers. Versioning and upgrading. https://www.stellar.org/developers/guides/concepts/versioning.html
11. Hard Forks and Soft Forks. What are they and what's their difference? https://medium.com/bitcoin-cryptocurrencies-and-blockchain-technology/hard-fork-and-soft-fork-what-are-they-and-whats-their-difference-5c00fab43e14

12. How Oracles connect Smart Contracts to the real world. https://medium.com/bethereum/how-oracles-connect-smart-contracts-to-the-real-world-a56d3ed6a507
13. Datta, A., Franklin, J., Garg, D., Jia, L. and Kaynar, D., 2010. On adversary models and compositional security. IEEE Security & Privacy, 9(3), pp.26-32.
14. Voigt, P. and Von dem Bussche, A., 2017. The EU General Data Protection Regulation (GDPR). A Practical Guide, 1st Ed., Cham: Springer International Publishing.
15. BitcoinWiki. Simplified Payment Verification. https://en.bitcoinwiki.org/wiki/Simplified_Payment_Verification
16. Kuhn, R., Information Technology-Role Based Access Control. ANSI Standard, Document Number: ANSI/INCITS, pp.359-2004.
17. Gunter, C.A., Liebovitz, D. and Malin, B., 2011. Experience-based access management: A life-cycle framework for identity and access management systems. IEEE security & privacy, 9(5), p.48.
18. TokenD XDR. https://docs.tokend.io/
19. Mazieres, D., 2015. The Stellar consensus protocol: A federated model for internet-level consensus. Stellar Development Foundation.
20. Callegati, F., Cerroni, W. and Ramilli, M., 2009. Man-in-the-Middle Attack to the HTTPS Protocol. IEEE Security & Privacy, 7(1), pp.78-81.
21. González, A.G., 2004. PayPal: the legal status of C2C payment systems. Computer law & security review, 20(4), pp.293-299.
22. Harn, L., 1994. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. IEE Proceedings-Computers and Digital Techniques, 141(5), pp.307-313.
23. Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system.
24. Kiayias, A., Russell, A., David, B. and Oliynykov, R., 2017, August. Ouroboros: A provably secure proof-of-stake blockchain protocol. In

Annual International Cryptology Conference (pp. 357-388). Springer, Cham.

25. Castro, M. and Liskov, B., 1999, February. Practical Byzantine fault tolerance. In OSDI (Vol. 99, pp. 173-186).
26. Libra. White paper. https://libra.org/en-US/white-paper/
27. Szabo, N., 1997. Formalizing and securing relationships on public networks. First Monday, 2(9).
28. Becker, G., 2008. Merkle signature schemes, merkle trees and their cryptanalysis. Ruhr-University Bochum, Tech. Rep.
29. Qiu, D. and Srikant, R., 2004, August. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In ACM SIGCOMM computer communication review (Vol. 34, No. 4, pp. 367-378). ACM.
30. Benet, J., 2014. Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561.
31. Ylonen, T. and Lonvick, C., 2005. The secure shell (SSH) protocol architecture (No. RFC 4251).
32. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. and Polk, W., 2008. RFC 5280: Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile. IETF, May.
33. Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S. and Adams, C., 2013. X. 509 Internet public key infrastructure online certificate status protocol-OCSP (No. RFC 6960).
34. Stephen Wilson, (1998) "Some limitations of web of trust models", Information Management & Computer Security, Vol. 6 Issue: 5, pp.218-220, https://doi.org/10.1108/09685229810240130
35. Nayak, G.N. and Samaddar, S.G., 2010, July. Different flavours of man-in-the-middle attack, consequences and feasible solutions. In 2010 3rd International Conference on Computer Science and Information Technology (Vol. 5, pp. 491-495). IEEE.

36. Freed, N. and Borenstein, N., 1996. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies (No. RFC 2045).
37. De Clercq, J., 2002, October. Single sign-on architectures. In International Conference on Infrastructure Security (pp. 40-58). Springer, Berlin, Heidelberg.
38. Zhang, Z.K., Cho, M.C.Y., Wang, C.W., Hsu, C.W., Chen, C.K. and Shieh, S., 2014, November. IoT security: ongoing challenges and research opportunities. In 2014 IEEE 7th international conference on service-oriented computing and applications (pp. 230-234). IEEE.
39. Wallet Server, Version Two: The Electric Boogaloo (The Tech Details). https://www.stellar.org/blog/wallet-server-version-two-the-electric-boogaloo-the-tech-details/
40. Simpson, W., 1996. RFC 1994: PPP challenge handshake authentication protocol (CHAP). Network Working Group, pp.1-12.
41. Neuman, C., Yu, T., Hartman, S. and Raeburn, K., 2005. RFC 4120: The Kerberos network authentication service (V5). Request for Comments.
42. Dacosta, I., Chakradeo, S., Ahamad, M. and Traynor, P., 2012. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. ACM Transactions on Internet Technology (TOIT), 12(1), p.1.
43. Chomsiri, T., 2008, November. Sniffing packets on LAN without ARP spoofing. In 2008 Third International Conference on Convergence and Hybrid Information Technology (Vol. 2, pp. 472-477). IEEE.
44. Ter Louw, M. and Venkatakrishnan, V.N., 2009, May. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In 2009 30th IEEE symposium on security and privacy (pp. 331-346). IEEE.
45. Alvarez, R.M., Hall, T.E. and Trechsel, A.H., 2009. Internet voting in comparative perspective: the case of Estonia. PS: Political Science & Politics, 42(3), pp.497-505.

46. Noether, S., 2015. Ring SIgnature Confidential Transactions for Monero. IACR Cryptology ePrint Archive, 2015, p.1098.
47. Boldyreva, A., 2003, January. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In International Workshop on Public Key Cryptography (pp. 31-46). Springer, Berlin, Heidelberg.
48. Maxwell, G., Poelstra, A., Seurin, Y. and Wuille, P., 2018. Simple schnorr multi-signatures with applications to bitcoin. Designs, Codes and Cryptography, pp.1-26.
49. Herlihy, M., 2018, July. Atomic cross-chain swaps. In Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (pp. 245-254). ACM.
50. Understand the GDPR in 10 minutes. https://medium.com/@ageitgey/understand-the-gdpr-in-10-minutes-407f4b54111f
51. Aumasson, J.P. and Endignoux, G., 2017. Gravity-sphincs. Tech. rep., National Institute of Standards and Technology.
52. Poon, J. and Dryja, T., 2016. The bitcoin lightning network: Scalable off-chain instant payments.
53. Chaum, D., 1983. Blind signatures for untraceable payments. In Advances in cryptology (pp. 199-203). Springer, Boston, MA.
54. Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D. and Simkin, M., 2016. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Public-Key Cryptography–PKC 2016 (pp. 301-330). Springer, Berlin, Heidelberg.
55. Andrew Poelstra. Blockstream Releases Confidential Assets. https://blockstream.com/2017/04/03/en-blockstream-releases-elements-confidential-assets/

Book

Pavel KRAVCHENKO
Bohdan SKRIABIN
Oleksandr KURBATOV

ENGINEER'S GUIDE TO FINANCIAL INTERNET

Authors' Test Edition

Editor
Bogdan Gryekhovodov

Cover Designer: Dmitrii Malakhov

Signed in the press on 30.06.2019.
The format is 60×90 1/16. Offset paper. Headset Times.
Usl. printing. sheet. 14. Accounting.-Ed. sheet. 12,32. View. No. 448.
Circulation 110 copies.
Printed in the printing house of "PROMART" Ltd
Tel. 38(057) 717-28-80

Financial Internet Manifesto

- Financial Internet is a set of independent systems that “speak” the same language.
- Technology ensures the freedom of choice of the asset users want to transact.
- Transaction validation is performed by the involved parties only.
- Transaction metadata (even the fact of its existence) can be private but can be proven by any involved party.
- Users control access to the assets they own.
- Financial Internet protocols and reference components should be a common good.

For an engineer, this means that:

- There is no single identity provider
- There is no global shared ledger
- There is no single “native” currency for transferring value between accounting systems
- All systems are independent, modular and can interoperate with other applications
- All systems should have a unified API to communicate with each other
- End users have access to a variety of applications (browsers) that all provide access to their digitized assets in any system
- All accounts are managed via cryptographic keys

For business people (if you are reading this), Financial Internet means free transactions, hassle-free accounting, easy onboarding for every person in the world, new business models, and no more papers.

"On the 8th of May, 2015, at the DBS blockchain hackathon in Singapore, Vladimir Dubinin and I came up with the vision for the future. Since then we have built and designed over 20 systems using the techniques described in this book. It became clear that ***all accounting systems in the world will be changed*** *in a way to communicate in the same language, and engineers will have A LOT of work to do. This book is an attempt to summarize our experience and help them to save time, money, and hassle."*

Pavel Kravchenko, Ph.D., Co-founder & CEO, Distributed Lab