Secret Smart Contracts in Hierarchical Blockchains

Sînică Alboaie  
Faculty of Computer Science, Alexandru Ioan Cuza University, Axiologic Research and RomSoft  
Iasi, Romania  
salboaie@gmail.com

Lenuţa Alboaie  
Faculty of Computer Science, Alexandru Ioan Cuza University  
Iasi, Romania  
adria@info.uaic.ro

Zeev Pritzker  
Arteevo  
Tel Aviv, Israel  
zeev@arteevo.com

Adrian Iftene  
Faculty of Computer Science, Alexandru Ioan Cuza University  
Iasi, Romania  
adiftene@info.uaic.ro

Abstract

This article presents the results of an implementation of a new platform based on swarm communication and executable choreographies. In our research of executable choreographies, we have come up with a more general model to implement smart contracts and a generic architecture of systems using hierarchical blockchain architecture. The novel concepts of secret smart contract and near-chain are introduced. The near-chain approach presents a new method to extend the hierarchical blockchain architecture and to improve performance, security and privacy characteristics of general blockchain-based systems. As such, we are subsequently defining and explaining why any extension of blockchain architectures should revolve around three essential dimensions: trustlessness, non-repudiation and tamper resistance. The hierarchical blockchain approach provides a novel perspective, as well as establishing off-chain storages (near-chains) as special types of hierarchical blockchains stored in a distributed file system. Furthermore, we are providing solutions to the difficult blockchain concerns regarding scalability, performance and privacy issues.

Keywords: Executable choreography, Swarm communication, Choreographic smart contract, Secret smart contracts, Time-locked smart contracts, Hierarchical blockchains, Data self-sovereignty.

1. Introduction

Blockchain technologies are contributing to transformative changes in business and society. The exponential increase in the Bitcoin price in the last five years created a massive wave of interest in research of blockchain related technologies such as the smart contracts [24]. Blockchain enables new types of innovation processes and offers opportunities for disintermediating network services resources. Blockchain technologies can also potentially increase transparency and citizens’ participation in democratic processes.

To create a real impact, this technological evolution must be accompanied by features that enhance trust and transparency. Guaranteeing transparency, trust, and respect for
privacy in a potentially hostile or untrusted network environment is, however, a non-trivial
task. Moreover, governance of such decentralized systems should be further studied and
better understood to extract the benefits of the technology while offsetting its risks.

We propose a new approach to implement smart contracts, called choreographic smart
contracts (CSC) that takes into consideration the complex relationship between technology
and society. The CSCs are designed to increase trust in blockchain based information
systems and provide technical means to address the relevant ethical considerations.

Smart contracts are computer protocols that facilitate, verify or enforce the negotiation
or performance of a contract. The term “smart contracts” was coined by computer scientist
Nick Szabo [24]. His purpose was to have contract law and related business practices
included in the design of e-commerce protocols between strangers on the Internet. Proponents of smart contracts claim that many kinds of contractual clauses may be made
partially or fully self-executing, self-enforcing or both. The aim of smart contracts is to
provide security that is superior to traditional contract law and to reduce other transaction
expenses associated with contracting [24].

Smart contracts have been primarily associated with cryptocurrencies. The most
prominent smart contract implementation is the Ethereum blockchain platform. The real-
world smart contract that gained mainstream coverage was the DAO - Decentralized
Autonomous Organization for venture capital funding, running on Ethereum that was
hacked and drained of approximately US$50 million in cryptocurrency three weeks later
[16].

Live Contract is a concept circling around for the same time. An example of a proper
implementation is Ricardian contracts [13]. A Ricardian contract compiles all information
from the legal document in a format executable by software. Smart contracts are therefore
a means of digital enforcement of an existing agreement that is executed automatically. A
Ricardian smart contract also contains (or refers to) the “legal prose” that describes the
intent of the contracting parties and the contractual. Using cryptographic hashes that refer
to external documents, Ricardian contracts can also refer to the code. There will
undoubtedly be more cross-pollination between Ricardian contracts and smart contracts in
the future and transactions will probably be carried out on the basis of different hybrid
contract forms.

Many of the problems of Ethereum [18] are caused by the fact that the contracts can
call each other but the whole execution is not atomic, thus having security issues [11] that
can be exploited by attackers. As we will see in the following chapters, in PrivateSky
project the participants can follow a much safer approach to implement smart contracts by
using a consensus algorithm that enforces atomicity of transactions for a whole chain of
smart contract invocations. By proposing a special architecture called “hierarchical
blockchain architecture” and by using the concept of executable choreographies we have
succeeded in proposing an alternative method to implement smart contracts and to improve
important aspects of the typical smart contracts implementation: performance, security,
and confidentiality.

2. Paradigms and Technologies Required to Understand PrivateSky Design
Choices

This section contains a short overview of technological bases that was developed in the
last years and it represents the technological background for our hierarchical blockchain
architecture proposal.

2.1. Swarm Communication Paradigm

In [4] authors proposed a new paradigm regarding communication mechanisms and
composability of services. In this model, called swarm communication, their perspective
on orchestration and choreography matches those presented in [20]: “Orchestration differs
from choreography in that it describes a process flow between services, controlled by a
single party. More collaborative, choreography tracks the sequence of messages involving
multiple parties, where no one party truly ‘owns’ the conversation.” Swarming, as it was defined, is different from service orchestration because there is no central controller that manages the business logic and there is no messaging sequence. In a swarm system, the individual nodes that represent the participant services do not fully control the answers to requests and they do not know the entire business logic.

The goal of swarm communication, as in the case of service integration patterns from SOA ([17], [22]) is to compose microservices/services’ behaviors. As we have seen in [4] swarm communications provide benefits of reducing application development efforts because it provides a natural environment to create and compose microservices. Microservice architectural style is an approach to develop applications as a suite of small services, each running in its own process. Microservices can be created independently by different teams with different technologies and can be tested in isolation. Redundancy and recovery from failures can more easily be implemented because the application is broken in small, independent processes. Heterogeneous systems integration and implementations of business processes and business rules can be performed by swarm communication between processes (called adapters) that are holding microservices.

In a swarm system, we distinguish three types of nodes: adapter nodes, client nodes, and infrastructure nodes. Adapter nodes are server-side nodes that provide some services or APIs for the existing applications (e.g. CRM, ERP, etc.). The nodes that offer system core functionalities are called infrastructure adapters. Client nodes are logically connected to an adapter by communication protocols created over TCP sockets or other protocols.

The communication mechanism is accomplished through the nodes “talk”: a node is talking to another node by sending a swarm. Sending a swarm means that it sends a message that is part of the set of swarm messages. In time, a swarm goes through a set of states (we called them phases) to accomplish a goal. Such phases contain code applications that change the internal state of the swarm or the state of the nodes in which the swarm got executed. It is important to notice that the phase’s code is not part of the node, but the code of the swarm itself, even if it is executed in the context of a node. Except for having a name (phase name) and a node hint, a phase is a behavior (code) that should happen (execute) when a message is received by a node.

In real systems, there are many communication patterns (e.g. request/reply, request/reaction, etc.), but in a swarm communication, complementary messages sent to solve a specific task can be automatically grouped in a simple representation which is the swarm itself. This aspect brings significant benefits regarding code maintenance and for the developer a comprehensible vision over the messages flow.

Because a swarm based architecture introduces only a few intuitive concepts like phases, adapters, groups, swarm description, and swarm primitives, it has a huge benefit for rapid learning [4]. Testability of real-world software systems is an important property and swarm-based service composability itself and also the services (adapters) can be easily tested together or in separation (by mocking real implementation of services) because existing tools and methodologies can be directly used by day one.

2.2. Executable Choreographies

Until 2015 the researchers have seen swarm communication only at the intra organizational level and the developed architectures (e.g. [10]) based on the Swarm paradigm functioned. The change of vision and the application of the swarms at the inter-organizational level has brought about changes both in vision and then in our implementations. Research and analysis have shown that swarm communication represents a good method for implementing executable choreographies [3], [5], [7].

The choreography is the formal description (non-executable choreography) or the concrete description (executable choreography) of a contractor process which involves the interaction of several systems belonging to departments or autonomous organizations, with different interests (e.g. regarding data privacy and security).

Usually, choreographies are viewed, as in the case of WSCDL (Web Service Choreography Description Language) [26] a method to describe formally contracts among
organizations.

In the academic environment [1], [8], [12], [14], [19] the concept of executable choreographies has been promoted. The idea was that these descriptions would be materialized by code (so no other description and even textual contracts are needed) to be executed by each organization that is part of the choreography. In our approach, the execution of choreographies can happen as business processes among multiple organizations or even inside a single organization between multiple security zones. In WS-CDL we only get a description of contracts. In order to use it in real systems, you have to transform choreographies descriptions in code, which implies supplementary programming effort. In the case of [14], while keeping influences from WS-CDL and BPMN, choreographies can be automatically projected onto a set of separate programs that can be therefore executed. In SwarmESB and later in the PrivateSky project authors have departed from the WS-CDL approach and they have found a way to implement executable choreographies using the swarm communication paradigm [5]. Using swarm communication there is no need to write supplementary code or to make a projection from the description of the choreographies into actual code.

Current research in the field comes with the introduction of three types of executable choreographies: Verified choreographies, Encrypted choreographies, and Serverless Choreographies [2], [5].

Verified choreographies have been naturally developed since 2014, following authors experience with SwarmESB. Encrypted and Serverless choreographies have been developed since 2016 and the authors claim that the implementation process in real projects is planned to be completed by 2021 within the PrivateSky project [21]. Verified Choreographies are executable choreographies accompanied by automated methods of verification in the usage of private data. Usage of private data can be observed only by checking the integration code (choreography) and this reduces the effort and the complexity of the verification instruments. Using choreography for integration leads to a logical separation between the code that runs in the processing nodes and the code that actually makes the integration.

Encrypted choreographies are based on encryption key control, mechanisms of identification and authentication enabling safe choreographies from the perspective of data protection between two or several organizations. The verifiable choreographies aim only at pinpoint the location (organizations) and the type of transferred private data. By contrast, the encrypted choreographies are making available a set of instruments and the self-implementation of choreographies minimizing the risk of sharing private information.

Serverless Choreographies are encrypted choreographies adapted to run platforms on a public cloud that provides full automation of deployment and monitoring. As there is no need for human intervention, we can increase the possibility of running cloud applications without access to private data by people with physical or administrative access to servers in the cloud. Basically, the serverless choreographies aim at enabling enterprise applications or mobile apps to use cloud resources almost at the same level of risk as if they were using a private server to which only the user has access to.

In summary, we can conclude that executable choreographies are actual code created to encode system behavior from a global point of view. The behavior of main entities in a system is given in a single program. Executable choreographies offer a concise view of the message flows enacted by a system. Choreographies enhance the quality of software, as they behave like executable blueprints of how communicating systems should behave. The choreography is both a legal agreement between independent or autonomous entities and an executable protocol that integrates the agreement securely within a digital infrastructure, meanwhile offering a high level of security because of cryptographic identification.

2.3. Blockchain Characteristics

Three essential characteristics make blockchain technologies attractive: trustlessness, non-repudiation and tamper resistance. These should be maintained by any extension or change of the blockchain architectures. Often presented as a method of eliminating the need for
trust, Blockchains actually does not completely eliminate trust in other participants in the system.

By trustlessness we mean the minimization of the level of trust required from a single actor in the system. This is done by distributing trust among the different players in the system through an economic game that stimulates actors to cooperate with protocol-defined rules. The network itself becomes the only point of trust. This network trust can actually translate through the need for trust in a majority of participants. An important element of truthfulness is the decentralization feature that has a topological aspect related to software architecture, but also a social or even political aspect. Decentralization is usually defined as the inability of political authority or corporation which can have total control over the system, thus preventing censorship. Representatives of a government, bank or other entity may not modify discretionary information in a decentralized application and may not force a particular behavior or opinion about the status of network participants’ applications. Despite the hype that assures the public of the decentralized nature of various cryptosystems, a more thorough technical analysis of decentralization must be a relative or gradual concept. A very sensitive aspect of decentralization is the way in which software is developed and distributed to the network participants. Without at least two-three completely independent implementations, perfect decentralization cannot exist. The degree of effective decentralization is hard to estimate objectively because any measurement of it would be influenced by a variety of factors, from psychology (the level of influence of charismatic leaders) to convoluted description and misunderstood weaknesses of technology. For example, estimating the level of centralization in Bitcoin or Ethereum (based on Proof of Work) shows a high level of centralization at the level of mining power. The difference between trustlessness and decentralization is quite subtle. Decentralization usually implies libertarian concepts that imply that states are detrimental to civil rights and freedoms. Some libertarians believe that delegation of trust to private companies controlled by market forces to be inevitable and acceptable. A perfect level of decentralization can be achieved in the topology of software architecture but from the point of view of the influence of the state, things are much more complicated. The opinion expressed in this article is that advanced governance mechanisms that improve the user experience and security of users’ data are desirable even if such mechanisms may not be compatible with the “decentralization” proposed by the libertarian way of thinking.

By non-repudiation, we understand that once a transaction is validated, it can no longer be contested by those who sign it. The person who signed a transaction must be in possession of the corresponding private key. The holder of the private key is solely responsible for protecting it. This can make blockchain systems hard to use for human users; people do not want unlimited responsibility. However, a system in which private keys can be easily stolen is not a system that we want to use. On the other hand, systems that attempt to limit user responsibility will compromise the trustlessness and decentralization properties. It is hard to imagine the employment on a very wide scale of systems that do not provide repair or revocation measures in the face of the actions of hackers or the inevitable human errors.

By tamper resistance, we mean how to ensure the integrity of transactions and their notarization. Transactions cannot be changed later, and the fictitious past transaction cannot be placed in the past even if an attacker has obtained the private keys. While practical applications of blockchain can compromise on trustlessness, decentralization, and non-repudiation, the tamper resistance cannot be compromised on. In fact, our definition of a blockchain based system necessarily includes tamper resistance. The rest of the features may need to be adjusted to make the systems usable.

An important aspect of tamper resistance is that even if an attacker successfully introduces invalid transactions into the system, sooner or later these transactions will be detected, and corrective measures will be taken (if the network governance allows this). From this point of view, the blockchain systems have opened a new dimension of computer security. Unlike the pre-blockchain systems, even if the attacks cannot be prevented, they will not remain unnoticed.
3. A Hierarchical Blockchain Architecture

PrivateSky implements a highly scalable blockchain-based platform that will provide a shared industry-wide foundation for implementing various applications, operational workflows, and business use cases. The underlying blockchain will be transparent to developers, enabling the systems' development by non-blockchain programmers. If additional smart contract functionality is required, the framework will provide a mechanism to deploy and integrate these into the platform. Pre-built business logic, configurable workflows, and built-in collaboration platform will allow all participants of the value chain to seamlessly communicate with trade partners, exchange data, implement contractual relationships and business processes and ensure compliance.

Blockchain-Based Hierarchical Architecture

The high-level PrivateSky reference architecture, shown in Figure 1, comprises four layers: Application, RESTful APIs, Near Chain Storage (off-chain) and Blockchain and Interchain layer.

![PrivateSky logical architecture - high-level vision](image)

*Fig.1. PrivateSky logical architecture - high-level vision [21].*

*Application layer* will contain the applications running on the PrivateSky platform. Applications will interact with the blockchain via *RESTful APIs*. Each of the applications will typically contain its own data layer (no-chain databases).

*Blockchain layer* is the core of the PrivateSky platform. It consists of multiple blockchain protocols to accommodate properly the different requirements from the application layer and to be flexible for new requirements and technologies. The various blockchain protocols/technologies interact and build a logical view of the PrivateSky Blockchain. Based on thorough research and analysis, existing high-performance open-source blockchain protocols, such as Hyperledger Fabric or others, can be adopted in the Blockchain Layer.

*Agents* uniquely represent the identities of the PrivateSky domain stakeholders (similar to accounts in the centralized internet servers). The globally unique identifiers of the accounts are based on the W3C standard Decentralized Identifiers (DID). Agents guarantee the authenticity of their transactions using robust digital signatures. Agents will execute highly flexible *self-sovereign* control of the *privacy* of their identity and Identity-Related Data (IRD; for example, an electronic health record or a home address) ([2], [9]). The current approach in PrivateSky is about using an identity management solution that has an open source implementation following the Data Model and Syntaxes for Decentralized
Identifiers (DIDs) [15].

Interchain layer contains pluggable middleware for interaction between the PrivateSky domain blockchain and external blockchains.

The hierarchy starts in a root domain (hub), contains any number of intermediate levels of satellite blockchains and leaves are CSBs (Cloud Safe Boxes - see below) or other external blockchains.

The PrivateSky platform does not propose a solution created around a single monolithic blockchain but a set of hierarchical blockchains called blockchain domains. As general vision, PrivateSky can be imagined as a blockchain composed of multiple smaller blockchain domains. In the root of the hierarchy, we could put an existing public blockchain (like Bitcoin blockchain) or a permission blockchain. The secret contract ideas imply the creation of even more satellite domains. It is feasible that a small blockchain domain to be deployed for a group of secret smart contracts between a small number of organizations. Therefore, PrivateSky could be envisioned as a world where decentralization means many independent but interoperable distributed applications: much more like web sites. We do not have the mindset of trying to build the perfect blockchain that should be used for all use cases. Each use case will require a customized solution. In line with this vision, each PrivateSky implementation and each blockchain domain will run customized software that evolves independently, exactly as an internet domain.

This approach could solve a series of existing issues with blockchains:

- **Solve scalability problems** as we can create any number of smaller blockchain domains;
- **Solve privacy-related problems** without reducing security: the whole data is still cryptographically fingerprinted within the root domain (indirectly but equally safe). With this approach, we will preserve the security benefit in having a big number of participants in the network but without sharing unnecessary data (or sharing every blockchain write operation) to every participant. Additionally, it provides confidentiality for scenarios where information may be deduced from simply observing the frequency that a party publishes transactions to a blockchain, even without knowing the contents (e.g. in supply chain use cases this could point to new product launches, shortages or recalls);
- **Removes limitations in adopting new innovations and changes** in technology as each
sub-domain can be upgraded and changed independently.

**Hierarchical Blockchains as a Single Source of Truth**

The hierarchical blockchain-based PrivateSky architecture, described above, provides a logical implementation of a decentralized database that serves as a single source of truth. Because satellite domains are periodically anchored cryptographically in the parent domains (their cryptographic fingerprint is saved at any change in the parent domain), the trustless, non-repudiation and tamper-proof properties of the whole hierarchical system are obtained. It is obvious from the mathematical properties that the data from a satellite blockchain-domain will have the same security level as data stored in an ideal blockchain replicated in as many nodes as all the satellites domains. To facilitate evolutionary digitization, straightforward integration of historical databases is supported by linking them to the blockchain. Some data can be stored on-chain, mainly in the form of immutable references to off-chain data such as cryptographic digital fingerprints. The bulk of the data will be stored off-chain to minimize costs, maximize blockchain performance, facilitate easy integration of historical databases, as well as containment of highly sensitive data. PrivateSky offers support for both on-chain and off-chain data to be encrypted using public key algorithms and be granularly access-controlled and content-addressable. The off-chain storage contains both data that is directly connected to the blockchain (it is anchored cryptographically in blockchain) and is referenced as near-chain storage as well as data stored in databases from the application layer (no-chain). Therefore we have two types of off-chain storages: the near-chain storages that can be also a special type of blockchain and no-chain storages that is represented by normal databases in the application layer.

**Near-Chain Storage with Cloud Safe Boxes**

The private or secret data storage in blockchain is not compliant to certain GDPR (EU General Data Protection Regulation) principles (such as “the right to be forgotten”) [2]. As such, only digital fingerprints of private data may be stored in the blockchain, allowing the verification of data authenticity and integrity of the near-chain data. These aspects are determining the CSB usefulness in secure data storage regarding GDPR as the entire CSB and all its past versions can be deleted when needed. CSBs are not a typical peer-to-peer blockchain that should stay online all the time but a new type of partially online blockchain that become “online” only for participants that can provide a decryption key.

The CSB (Cloud Safe Boxes) concept proposed by PrivateSky can provide a standard method to export and handle private or secret data from new or existing applications in order to enforce GDPR requirements and to provide a universal method of sharing confidential and encrypted data between people and organizations. If needed, the CSB technology can be used with other blockchains, but the programming model for the secret smart contracts technology is part of the PrivateSky.

The CSB concept aims at achieving the data self-sovereignty vision ([2], [10]): the legal owner of data should have total control on how its data is used by technical methods (and therefore not only by cumbersome legal recourse means). Revocation of consent for the confidential data exported in CSBs will be easily detectable by data processors that have no recourse against removing immediately all their copies.

A CSB is represented physically as an encrypted folder. Regular CSBs store data as PrivateSky specific assets (JSON following PrivateSky specific schemas compatible with JSON-LD [25]) and unstructured files. Regular CSBs are basically like small offline blockchains with special access rules about who can update or even delete them. On the other hand, these embedded blockchains are queryable as NoSQL databases, offering a concept of URL for all data stored inside and, as a result, behave as a new type of serverless database. The encrypted CSB files can be stored using regular cloud services (like S3, Google Drive, IPFS, Dropbox, etc) or with custom web service of PrivateSky open source implementation that was called Encrypted Distributed File System (EDFS). EDFS provides additional benefits regarding authorization or quota control and it is a recommended method for consortium controlled distributed ledgers developed with PrivateSky. Therefore, the CSB concept represents secure storage and sharing data system, designed
for private data management requiring high-security levels. The CSB was created as a system complementary to usual peer-to-peer blockchains, for decentralized private data storage.

A single CSB is identifiable in various ways that will not be presented in this article. The major takeaway is that the rules for identifications are aimed at improving the user experience regarding the handling of the encrypted data. Therefore, the central concept is that each CSB has a cryptographic identifier (called SEED). The SEED, when is shared in clear text or encrypted, it is equivalent to the encrypted content of the CSB. On this perspective, CSBs can contain references to other CSBs and are constituting an EDFS distributed file system, where each CSB is at the same time a folder (may contain internally attached files or references to other CSBs) and an embedded blockchain. Each CSB incorporates a blockchain containing assets and attached smart contracts that may trigger new transactions upon the assets, automatically or by user demand.

These characteristics make the CSB an ideal mechanism to store, share and monitor the sharing according to definite rules, secret information or private data. The inspiration and functionality of CSBs originated in the Privacy by Design principles and the European legislation (GDPR), provides a solid implementation mechanism of the data self-sovereignty principle.

4. Choreographic Smart Contracts and Secret Smart Contracts

To highlight the difference between the PrivateSky approach and the other approaches to smart contracts we have introduced the term choreographic smart contract. A choreographic smart contract is implemented as serverless choreographies, but can also be understood as having a more general form of validation compared to the classic smart contract. Thus, a classic smart contract (e.g. in Ethereum) certifies the validity of the transaction in all the validating nodes. A transaction is valid if it has all the digital signatures necessary and the side effects of the transaction are obtained only from code that is anchored in the blockchain. For example, paying an amount of cryptocurrency from an account A to an account B requires the signature of the account holder A, but at the same time, the transaction is valid only if the amount left by the account A is equal to the amount that reaches the B account.

Typically, the method to implement this validation requires the checking of signatures in all nodes and the execution of the transaction code in all nodes. This method is suitable for payment systems in cryptocurrencies but it is excessive for many other cases where the responsibility to validate a transaction can be delegated by the constitution of the blockchain (the collection of all smart contracts anchored in blockchain) to a smaller number of nodes, and thus the signatures on the result of the execution or other constraints are sufficient for validation. Exploring acceptable constraints to perform transaction validation without running the code in all nodes has led us to the concept of choreographic smart contracts.

The participating nodes have different access to secrets and thus justify the need to use executable choreographies that do validation in multiple phases. Therefore, choreographic smart contracts bring obvious performance and confidentiality advantages. When the benefits of choreographic smart contracts are about confidentiality and not performance, we talk about the concept of secret smart contracts. The secret smart contracts, support computation with the blockchain validator nodes being blind to the input data as well as to the results of the computation while retaining the ability to validate them according to the constitution of the blockchain.

The execution of choreographic smart contracts requires complex choreographies between multiple agents in order to accept changes of states in the blockchain serialization. A smart contract execution environment ensures that all nodes are validating independently the execution of the smart contract and are rejecting those executions that do not follow the contract rules.
From the confidentiality perspective, in Figure 3, we have identified the following layers:

- **The Self-Sovereign Storage Layer (S-SSL).** Self-Sovereign Storage Layer will be based on the Cloud Safe Boxes concept. CSBs offer shared control over bilateral facts and their evolution. This layer encrypts information while distributing keys to selected authorities or participants who can access it. This way, the software allows permission to clients to create and manage cryptographic secrets. A proxy understanding of the S-SSL is to imagine an entire internet-wide storage system having security properties similar to the data stored in cryptocurrency wallets.

- **The Smart Contracts Layer (SCL) and the Secret Smart Contracts Layer (SSCL).** The SCL layer offers a similar concept of smart contract comparable with other blockchains. SMCL layer offers various types of secret smart contract and encrypted choreographies. This layer allows the encryption of both the storage and the smart contract actual code in order to ensure the privacy of any transaction, under the assumption that the authorizers can be trusted not to collude. The SMCL provides a trusted execution environment for DApp developers to spin up secret smart contracts.

- **Multi-Dimension Locked Secrets Layer (MDLSL).** A time-lock is a way of securing a file or message until a certain time in the future. It is an important concept for whistleblowers, document embargoes and blind auctions. Time-locking in practice has been difficult, especially on public blockchains like Ethereum. If users try to cheat the system, by falsifying data or publishing it too early, they will be penalized. Going forward, we can theorize that incentive schemes could be extended, so that time is not the only variable in releasing a secret.

These layers necessary in implementing smart secret contracts (and in particular the Self-Sovereign Storage Layer) lead to our conclusion that there is a need of a distributed file system or an improved decentralized protocol of services to store and share encryption keys. This article proposes such a potential system in the form of the Cloud Safe Boxes and its future improvements.

5. **Conclusions and Future Work**

In this article we have presented a hierarchical architecture that allows the integration of many different blockchain technologies into a tree hierarchy that has in the leaves a special type of blockchain called near-chain and represented by Cloud Safe Boxes. The idea of a secret smart contract implemented through swarm communication and choreographic smart contracts is presented to the research community as a novel approach. We are actively working on methods of using homomorphic encryption [6] and multi-party computation (MPC) [23] and distributed storages [9] to implement an execution engine for secret smart contracts by doing computation on encrypted data. This work is aimed at extending the current CSB implementation in order to support multidimensional locked...
We are confident that the choreographic smart contracts will continue to open new research directions with great potential and major impact on the future of the Information Systems Development.

Acknowledgments

This work is partially supported by POC-A1-A1.2.3-G-2015 program, as part of the PrivateSky project (P_40_371/13/01.09.2016) and the Horizon 2020 project WeldGalaxy (Grant agreement 822106/2018).

References


