



DS-06-2017: Cybersecurity PPP: Cryptography


PRIViLEDGE
Privacy-Enhancing Cryptography in Distributed Ledgers

**D4.2 – Report on Architecture for
Privacy-Preserving Applications on Ledgers**

Due date of deliverable: 30 June 2020
Actual submission date: 30 June 2020

Grant agreement number: 780477
Start date of project: 1 January 2018
Revision 1.0

Lead contractor: GRNET
Duration: 36 months

 Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020	
Dissemination Level	
PU = Public, fully open	X
CO = Confidential, restricted under conditions set out in the Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC	

D4.2

Report on Architecture for Privacy-Preserving Applications on Ledgers

Editor

Panos Louridas (GRNET)

Contributors

Athina Kleinaki (GRNET)
Mikhail Volkhov (UEDIN)
Markulf Kolfweiss (UEDIN)
Georgios Anastasiou (GUNET)
Sergei Kuštšenko (SCCEIV)
Sven Heiberg (SCCEIV)
Ivan Visconti (UNISA)
Vincenzo Iovino (UNISA)
Vincenzo Botta (UNISA)
Ahto Truu (GT)

Reviewers

Nikos Karagiannidis (IOHK)
Ahto Truu (GT)

30 June 2020

Revision 1.0

The work described in this document has been conducted within the project PRIViLEDGE, started in January 2018. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780477.

The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

©Copyright by the PRIViLEDGE Consortium

Executive Summary

This document presents the high-level architecture that will be followed in the implementation of three of the PRIViLEDGE use cases: verifiable voting, health insurance, and digital credentials. Moreover, it presents the architecture and the goals of PRIViLEDGE's horizontally cross-cutting toolkits that provide functionality that could be used in different situations to provide the functionality for zero-knowledge proofs in ledgers; secure two-party and multi-party computations on ledgers; and privacy-preserving data storage on ledgers. The purpose of the document is to act as a guide for the implementation of the use cases and the toolkits in the remainder of the project. The detailed architecture of the use cases and the toolkits will of course evolve as implementation progresses and feedback is gathered, both regarding requirements and regarding technological developments.

Contents

1	Introduction	1
2	UC1: Verifiable Online Voting with Ledgers	3
2.1	Architectural Goals	3
2.2	Election and Voting Methods	3
2.2.1	Roles in an Election	3
2.2.2	Election Stages	4
2.3	Architecturally Significant Requirements	5
2.4	System Components	6
2.4.1	Election Management Service	7
2.4.2	Bulletin Board	7
2.4.3	Voting Application	8
2.4.4	Vote Collector Service	8
2.4.5	Key Management Tool	8
2.5	High-level Protocol	8
2.5.1	Stage: Election setup	8
	Trust Root Initialization	8
	TLS Key Generation	9
	Signature Key-Pair Generation	9
	Bulletin Board Initialization	9
	Encryption Scheme Key-Pair Generation	9
	Voter List	9
	Election Configuration	9
	Election Data Signing	9
	Technical Configuration	10
	Configuration Export to Instances	10
	Election Setup Summary	10
2.5.2	Stage: Voting	11
	Voter Authentication	11
	Election Data Retrieval	11
	Marking Choices	11
	Complete Ballot Encryption	11
	Verifying the Public Record	12
	Vote Submission Summary	12
2.5.3	Stage: Tally	14
	Exporting the Ballot Box	14
	Ballot Box Processing	14
	Results Decryption	14
	Tally Phase Summary	15

2.5.4	Stage: Audit	15
	Results publication	15
	Audit Phase Summary	16
2.6	Architectural Mechanisms	16
3	UC2: Distributed Ledger for Health Insurance	18
3.1	Introduction	18
3.2	Requirements	18
3.3	High-Level Architecture	19
3.4	Data Model	20
3.5	Component Interactions	20
3.5.1	Updating Patient Records	20
3.5.2	Reviewing Patient Records	21
3.5.3	Compilation of Reports	21
3.6	Cryptographic Tools	21
3.6.1	Commitment Scheme	21
3.6.2	Multi-Party Computation Protocol	22
4	UC3: University Diploma Record Ledger—The DIPLOMATA System	23
4.1	Introduction	23
4.2	Requirements	23
4.3	Approach	24
4.3.1	Overview	24
4.3.2	The DIPLOMATA Protocol	24
	Preliminaries	24
	Protocol Steps	25
	Addressing Possible Attacks	26
4.4	Architectural Considerations	26
5	Toolkit for Zero-Knowledge Proofs in Ledgers	28
5.1	SNARK Setups	28
5.2	Subversion-Resistant Zero-Knowledge	28
5.3	Soundness with Updateable Setup	29
5.4	Private Smart Contracts and zkay	29
5.5	Practical Private Smart Contracts with Updatable SRS	31
5.6	Future steps and extensions	33
6	Toolkit for Ledger-Oriented Secure Two/Multi-Party Computation	35
6.1	Module 1: Abstract and Safe Communication with a Blockchain	35
6.2	Module 2: Verifiable Honest-but-Curious Secure Multi-Party Computation	36
7	Toolkit for Privacy-Preserving Data Storage on Ledgers	38
7.1	Cryptographic Techniques for Security and Privacy in Ledgers	38
7.2	Safe Storage of Private Data Used on Ledgers	40
7.3	Privacy-Preserving Ledger Storage Toolkit	42
8	Conclusions	46

Chapter 1

Introduction

This document describes the architecture for privacy-preserving applications on ledgers for application domains not covered by Deliverable D4.1, “First Report on Architecture of Secure Ledger Systems”. It shows how Distributed Ledger Technologies (DLTs) can be leveraged to tackle privacy-related problems at two different levels: that of particular use cases, and that of horizontal concerns that emerge in different settings.

Starting with the use cases, the deliverable covers three of the PRIViLEDGE use cases. For each one of them it outlines the problem, the requirements and overall goals, and then the approach that will be followed to implement a solution using ledger technologies. The description of the approaches is at a relatively high, architectural level, as the implementation will be fleshed out in the remainder of the project.

Chapter 2 (UC1: Verifiable Online Voting with Ledgers) deals with the use of ledgers to improve privacy guarantees in online voting. The proposed solution uses a protocol that makes it possible to prove to the independent auditor in a voter privacy preserving manner that all accepted votes were stored, sent to the tabulation according to the election rules, and decrypted/tabulated correctly.

Chapter 3 (UC2: Distributed Ledger for Health Insurance) provides the means for a “pay for performance” insurance model, where the payments that providers receive depend on achieving certain measurable outcomes, such as reducing the mean time off work for a certain diagnosis or lowering the number of patients with blood pressure exceeding a given threshold. For this to be feasible, it is necessary that all participating health care providers share the same view of patient records, that patient records are kept private, that the integrity of medical histories is not compromised, that care providers are accountable for records they pass among themselves, and that reported metrics are consistent. Ledger technologies will be used to meet these requirements.

Chapter 4 (UC3: University Diploma Record Ledger) presents a way to implement a verifiable credentials model pertaining to digital certificates, via a cryptographic protocol that commits the workflow on a ledger. The protocol is built around the idea that a degree holder wants to prove possession of a degree, awarded by an issuer (e.g., a university) to a verifier. By using the described protocol, no party can later deny that proof of the degree has been presented to the verifier; moreover, the verifier cannot leak information about the degree per se without compromising the verifier’s own privacy.

The next three chapters (5–7) present three different generic toolkits that can be used to enhance privacy in applications using ledger technologies.

Chapter 5 will implement technologies facilitating zero-knowledge proofs on ledgers. Although there has been considerable work in the area, many zero-knowledge proof constructs rely on a Common Reference String (CRS) that, in order for the proofs to be secure, must be constructed in a specific way. Currently, one way is to use a public ceremony for that, but such an approach may not be practical or efficient in different circumstances. PRIViLEDGE will investigate and implement an alternative approach, whereby the CRS can be updated for new uses, without compromising security or privacy.

Chapter 6 will implement technologies that combine blockchain and Multi-Party Computation (MPC). Two modules will be developed. The first one will abstract the use of a blockchain allowing to securely run on a generic blockchain protocol secure two and MPC that were originally designed to run on point-to-point chan-

D4.2 – Report on Architecture for Privacy-Preserving Applications on Ledgers

nels. The second will extend a library for honest-but-curious secure MPC in order to add verifiability of correct computation through zk-SNARKs.

Chapter 7 gives an overview of techniques for privacy and security of the data and the transactions in the ledgers themselves and techniques for the safe storage of private data by ledger applications; before presenting the architecture of a toolkit for privacy-preserving data storage on distributed ledgers.

Chapter 2

UC1: Verifiable Online Voting with Ledgers

2.1 Architectural Goals

The objective of Tiviledge is to provide secure, usable and transparent online voting by using a protocol that makes it possible to prove to the independent auditor in a voter privacy preserving manner that all accepted votes were stored, sent to the tabulation according to the election rules, and decrypted/tabulated correctly.

Tiviledge uses the notion of the data-audit to ensure that the published voting result corresponds to encrypted preferences sent by eligible voters to the digital ballot box and the bulletin board. We mitigate the need to prove correctness of the software and its operation by demonstrating that according to the public protocol, the correct election outcome was calculated based on the given public inputs.

Moreover, we emphasize the importance of long-term voter privacy over the long-term integrity of the election result.

2.2 Election and Voting Methods

An election is a process by which a population makes a political decision / elects an individual into office. An election relies on a single electoral system to determine the means by which votes are translated into seats. An electoral system combines the preferences of voters to a decision, according to predefined rules. An election relies on one or many voting methods to determine the means by which preferences are gathered from the eligible voters. The elections that we are concerned with shall be free, general, uniform, and direct. The ballot is secret.

Since an electoral system combines different preferences of many voters into a single decision, the transparency and integrity of an election becomes important—the losers must be able to believe that they lost because of lack of support and not because of the manipulation of results.

The general requirements of transparency and integrity are in conflict with the requirement of ballot secrecy. The latter is a mechanism to ensure the freedom of vote. Elections without ballot secrecy are only possible in an environment without bribery and coercion.

2.2.1 Roles in an Election

There are following major roles in the election process:

- *Voter*—someone eligible to participate in the decision making process by casting her preference as a vote;
- *Candidate*—an entity in the decision making process, running for the position under question;
- *Election Official*—someone responsible for organizing and actively carrying out the decision making process, in order to determine the decision.

Additionally, there is a role of an *Auditor*—someone who has the right to participate in the decision making process as an observer with access to procedures and artifacts necessary for ensuring proper conduct. In paper-based voting in polling stations, the role of an Auditor is to observe the voting process and the handling of the physical artifacts created during the process.

In electronic voting many critical steps of the voting process cannot be directly observed by a human being. In particular, in electronic voting in a remote setting it becomes important to audit election based on the election data: we need protocols that allow auditors to verify that the election result was correctly calculated based on the votes that were gathered from eligible voters during the voting period.

We note that some roles can be shared by one individual, but in some cases there are role conflicts. For example, although Election Officials themselves may carry out exactly the same tasks as Auditors, they are not considered Auditors themselves. Independence from election organizers is expected from the Auditors. Another role conflict is that of Candidates and Election Officials—a Candidate cannot be an Election Official at the same time.

2.2.2 Election Stages

We divide the election process into three stages: election setup, vote submission, and tally.

The election setup stage includes the following general duties:

- The Election Officials prepare the list of Candidates;
- The Election Officials prepare the list of eligible Voters;
- The Election Officials purchase technology for implementing the voting process;
- The Election Officials prepare the voting technology and procedures for the election;
- The Auditor actively participates in the system initialization;
- Election Officials publish all relevant information to general public.

The vote submission phase includes the following duties:

- The Voter uses a voting tool to cast her will as a vote;
- The Vote is transported to the ballot box and a bulletin board under control of Election Officials and Auditors;
- The Election Officials and Auditors accept the vote to ballot box and bulletin board based on Voter eligibility;
- The Election Officials and Auditors store the vote in the ballot box and on the bulletin board all through the voting period;
- The Auditor verifies the transported vote to the bulletin board.

Finally, the tally phase must support the following duties:

- The Election Officials open the ballot box and use a tabulation tool to determine the result and publish it to the bulletin board for further verification;
- Any observer can access the bulletin board for tally verification.

We note that an election cannot take place without a technology consisting at least of an Election Preparation Tool, a Voting Tool, an Eligibility Verification Tool, a Ballot Box, a Tabulation Tool and a Bulletin Board. The above duties and these tools must be reflected in the architecture, their implementation is governed by underlying technology and architecturally significant requirements.

2.3 Architecturally Significant Requirements

We now address the main requirements that influence the architecture of the Tivledge voting system.

Requirements Related to Voting and Eligibility Verification

- *ballot-structure-simple*: A Voter can post exactly 1 choice from set of n candidates as a vote to a ballot box;
- *ballot-structure*: A Voter can post up to m unordered choices from set of n candidates as a vote to a ballot box;
- *eligibility-A*: Voters from set of eligible people can post votes to a ballot box and bulletin board;
- *eligibility-B*: entities not in set of eligible people are blocked from posting votes to a ballot box and bulletin board;
- *bullet-box-authenticity*: A Voter can verify that the person she is communicating with is indeed the Election Official;
- *limited-time*: The voter can only post a vote to a ballot box and bulletin board during a specified time (the voting period);
- *re-voting*: A Voter can post multiple votes to a ballot-box and bulletin board during the voting period.

Voter Verifiability

- *cast-as-intended-A*: The voter can verify that the vote to be posted correctly reflects her choices from a set of candidates;
- *cast-as-intended-B*: The voter can verify that the posted vote correctly reflects her choices from a set of candidates;
- *recorded-as-cast*: The voter gets a irrefutable/verifiable confirmation for a posted vote that it was stored in the ballot box and the bulletin board irrevocably, in its original form;
- *timeliness*: The Voter gets the recorded-as-cast confirmation in timely manner.

We note that *cast-as-intended-A* reflects methods such as Benaloh challenge [Ben07]. Tivledge shall implement the *cast-as-intended-B* which reflects methods such as Estonian style verification with different device [HW14].

Long Term Vote Storage and Tally

- *persistence*: All posted votes are available for tallying after the end of the voting period;
- *persistence-malicious*: All posted votes are available for tallying after the end of the voting period under the assumption the voting system was partially under control of an adversary;
- *no-partial-result*: The tally is not published before the end of a voting period;
- *uniformity*: Only one of voter's votes will be taken into account for the tally¹;

¹Usually it is the last vote, however there are schemes such as JCJ [JCJ05] that apply other strategies for taking that decision.

- *well-formedness*: Only votes that contained up to m unordered choices from set of n candidates are included in the tally;
- *revokability*: Votes can be excluded from the tally upon an authorized request by Election Officials;
- *tally*: The tally can be published after the end of a voting period.

Auditor Verifiability

- *eligibility-verification*: The Auditor can verify that the ballot box and the bulletin board contained votes only from eligible voters;
- *integrity-persistence-verification*: The auditor can verify that the set of posted votes sent for tallying is the same as the set of posted votes accepted from the eligible voters;
- *post-processing-verification*: The Auditor can verify that votes excluded from the tally were revoked on purpose;
- *tally-verification*: The Auditor can verify that the set of posted votes yields the tally published;
- *privacy-preserving-audit*: The Auditor does not learn which choices a particular voter made for any of the votes posted.

We note that the initial duty of auditing an election lies within the Election Officials themselves. Nevertheless, these requirements must be met in case of an auditor who is not part of the election organization.

Voter Privacy In the following list, X may be Election Official, Auditor or any component of the Voting System / Technology.

- *voter-privacy-A*: The X does not leak which choices a particular voter made for any of the votes posted;
- *voter-privacy-B*: The X does not learn which choices a particular voter made for any of the votes posted;
- *voter-privacy-C*: The X is incapable of learning which choices a particular voter made for any of the votes posted;
- *receipt-freeness*: A Voter cannot prove to a third party that her particular vote was tallied as recorded.

In Tivledge we aim for *voter-privacy-B* towards Election Official and *voter-privacy-C* towards Auditor, meaning that if a malicious Election Organizer might be in the position to find out who voted for whom, then a malicious Auditor shall not be capable of making the same connection despite the data available to him. This target is justified by the desire to allow anyone to audit the correctness of a voting result with respect to the contents of the digital ballot box and set of eligible voters.

2.4 System Components

Tivledge consists of different parts where each part is responsible for specific actions. Figure 2.1 shows the main components of the desired system.

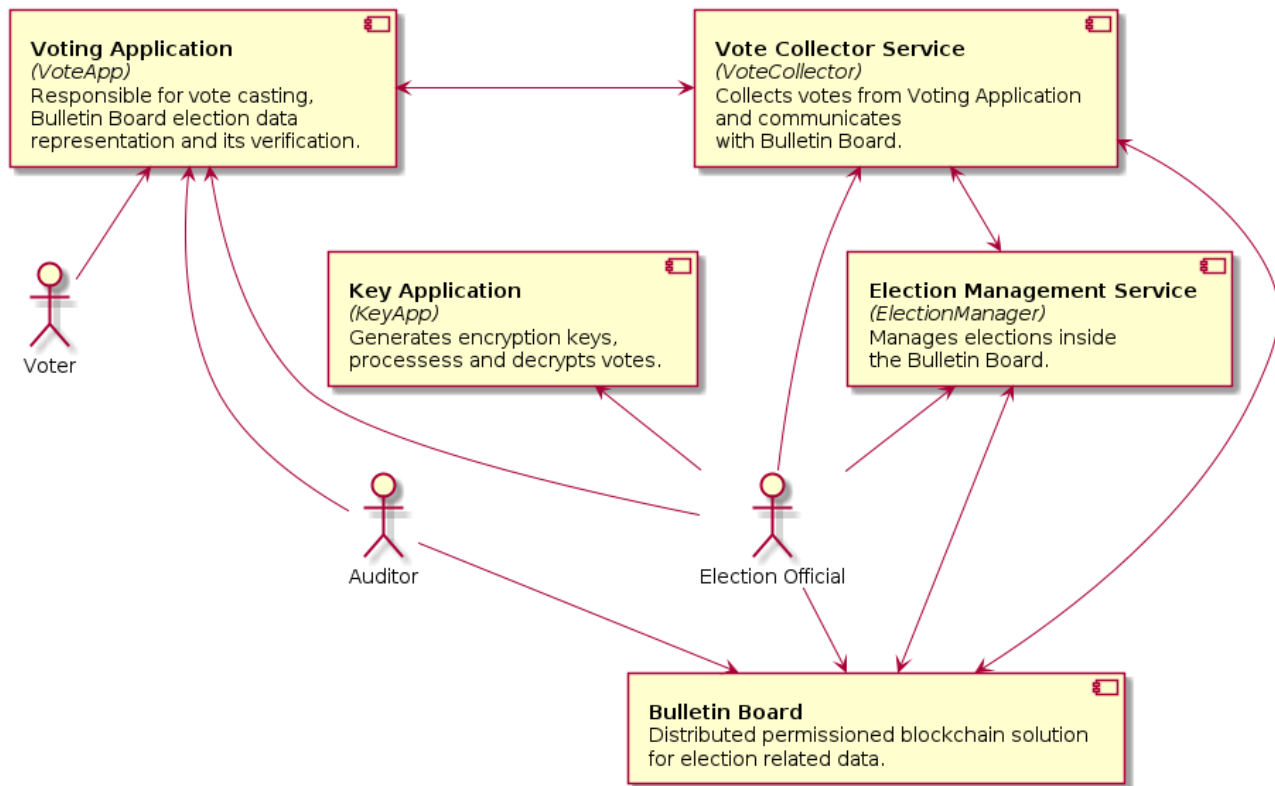


Figure 2.1: Tiviledge Main Components.

2.4.1 Election Management Service

It is used by the Election Official for different election stages and procedures. The main responsibilities of this service are:

- election addition and deletion;
- voter list publication;
- revocation list publication;
- ballot box retrieval from the Vote Collector Service.
- tally results publication;

2.4.2 Bulletin Board

The Bulletin Board serves as a public and trusted source of election data, which is available to anyone. The only restriction is that anyone can access the data on the bulletin board, but only permissioned parties have access to add and modify the data inside the bulletin board.

The publicly available data consists of:

- election configuration;
- voter list;
- revocation list;

- vote public part;
- election results.

The Bulletin Board will be implemented as a permissioned distributed blockchain network with the help of HyperLedger Fabric.

2.4.3 Voting Application

The Voting Application offers the functionality for casting a ballot and verifying different election steps, where verifications are made on the Bulletin Board's data:

- Verification of the published public part of the vote:
 - voter eligibility check;
 - ballot well-formedness proofs verification.
- Signature verification of election configuration, voter list, revocation list, election results published by the Election Official;
- Tally results verification.

The Voting Application is used as well by the Auditor and any observer in addition to Voters. The reason that Auditor uses the same application for verification as the Voter is to increase trust between Election Organizers and Voters. If the same verifications satisfy the Auditors, then the Voters can be sure that the election is honest.

2.4.4 Vote Collector Service

The main responsibility of this service is to form ballots with the Voting Application and store them locally in the ballot box. Moreover, the Vote Collector Service serves as a proxy between the Bulletin Board and the Voting Application, where it transports data from the Bulletin Board to the Voting Application.

2.4.5 Key Management Tool

The Key management Tool is used by the Election Official for encryption and decryption keys generation, as well as for votes aggregation.

2.5 High-level Protocol

The Tivledge protocol relies on Commitment Consistent Encryption (CCE) to construct a perfectly private audit trail, as specified in [Per]. Tivledge implements the receipt-freeness property as proposed by [Avi19]. The protocol is executed in following stages: election setup, vote submission, tally, audit.

2.5.1 Stage: Election setup

The first step would be to set up the necessary cryptographic material for different services and applications that will then proceed to other initialization steps.

Trust Root Initialization

The trust root is a signing key-pair that is used to sign a public key to denote that it is issued by a trusted authority. Anyone verifying the signed public key can construct a trust chain up to the trust root. Using a trust root, the system is only dependent on the root instead of a set of fixed public keys. The trust root also contains the public key for verifying voter credentials.

TLS Key Generation

The parties and the components in the voting system communicate with each other over TLS. To be able to ensure authenticity and secrecy using TLS, voting system parties use client- and server-authenticated mode of TLS. In this mode, the connection endpoints can verify that they are indeed communicating with the expected party.

To be able to provide client- and server-authenticated mode of TLS, the public keys used for TLS have to be signed by the trust root. The trust root also provides the identity to the used public key. In practice, the public keys and trust root signatures are encoded in X.509 certificate.

The public-facing server gets additionally a public key signed by a widely recognized trust authority.

Signature Key-Pair Generation

In addition to keys for using together with TLS, a set of keys will be generated to sign different requests by backend components. During protocol run, when a request is signed by a party, then the corresponding party can not later deny making the request.

A signature key-pair is generated for signing requests for vote publication to the Bulletin Board and its storage in the Vote Collector Service (ballot box) as well for signing successful authentication requests and for signing static configuration for voters.

The signature key-pairs are not signed by the trust root as they are fixed throughout the protocol run and signed within the election configuration.

Bulletin Board Initialization

The Election Official with the Auditor set a up a Bulletin Board, they both agree on who can add, read and modify information inside the Bulletin Board and under what conditions.

Encryption Scheme Key-Pair Generation

The key generation function depends on used key management technology. It is recommended that the key is generated by using smart cards and threshold scheme without trusted dealer. The public key is exported.

Voter List

The Election Official creates a voter list that specifies who can participate in the upcoming election. The electorate roll has to be a list of unique identifiers, one for every voter.

Election Configuration

From a voting protocol view, the important configuration variables are the contest identifier, contest descriptive name, candidate list, encryption public key, the contest open time and the contest closing time.

Election Data Signing

The election data consists of the contest configuration, voters list, candidate list, and revocation list. The goal of signing is to defend against serving incorrect election data to the voter and replay attacks. By serving incorrect data to the voter the results of the election can be incorrect even if all parties are behaving correctly. Therefore signed election data can be later exported to different applications as they represent valid and trusted data.

Technical Configuration

The Election Official creates different configurations for applications and services that contain previously generated cryptographic material and specifies additional parameters for each component.

Configuration Export to Instances

The election data is transferred to the voting system participants. As this step is done during the election setup, there exist still no trusted channels between the participants and thus the configuration has to be exported in a previously bootstrapped channel. In practice, the machines where the configurations are exported to are in physical control of the election committee and thus the communication channels can be trusted due to physical assumptions. Different system components are then initialized with imported configurations.

Election Setup Summary

The previously described steps are summarized in Figure 2.2.

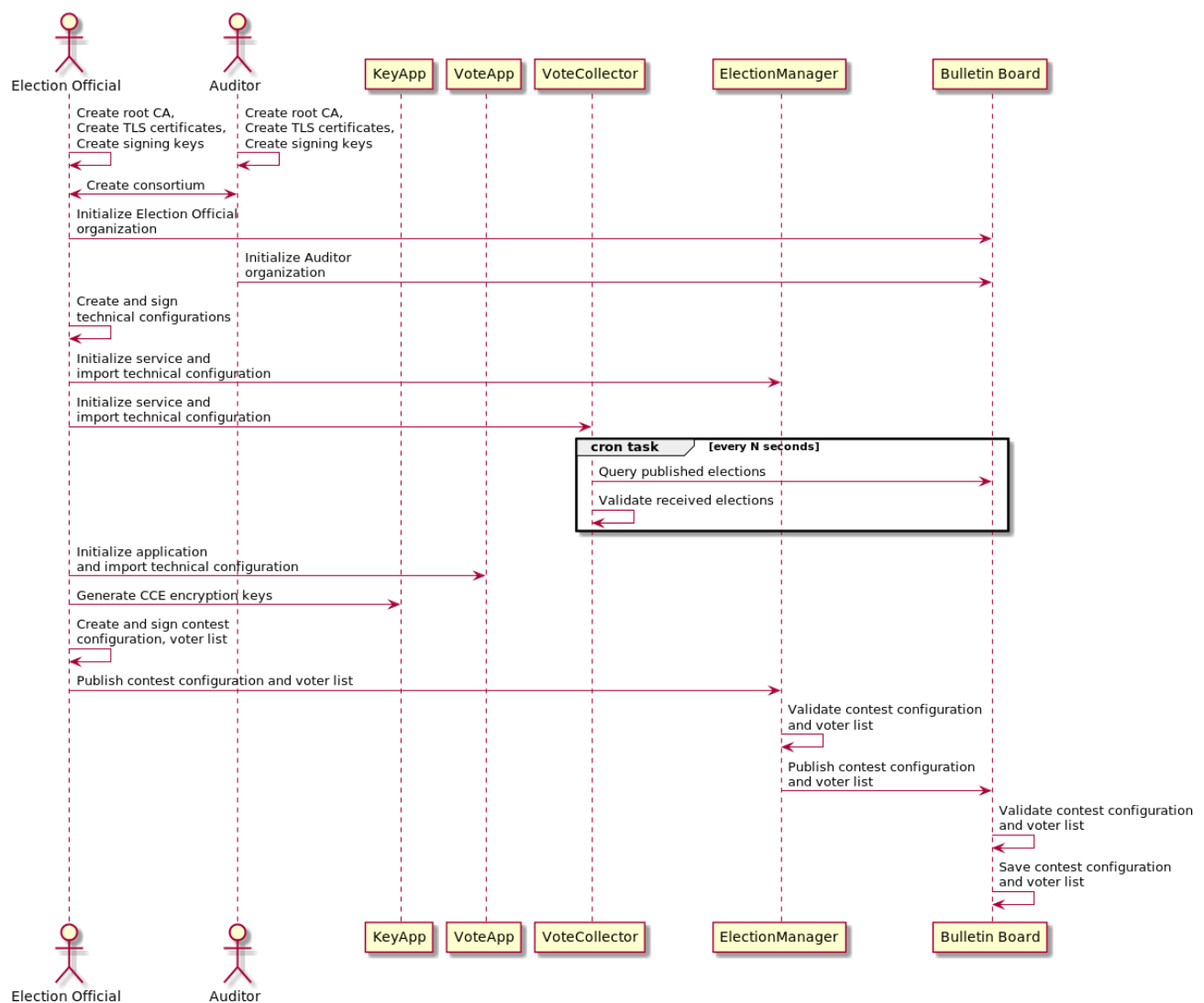


Figure 2.2: Election setup summarized sequence diagram.

2.5.2 Stage: Voting

In the vote submission phase, voters use the VoteApp to interact with the VoteCollector and the Bulletin Board for verifiable vote casting.

Voting consists of authentication, election data retrieval, choice marking, ballot encryption, vote addition to storage and publication to bulletin board.

Voter Authentication

The Voter authenticates themselves to the VoteApp by providing their certificate to the VoteCollector to receive elections to which they can participate. When the VoteCollector receives a voter's certificate it checks that the certificate was issued from the root certificate in the election configuration. If this verifies, VoteCollector makes sure that this particular voter exists in the voter list and does not exist in revocation list. If all verifies this election is marked as available for the voter and sent to the VoteApp.

Election Data Retrieval

After successful authentication to the VoteApp, the available elections are received from the VoteCollector.

Then the Voter sees the different elections which are available to them. Elections are provided with their descriptive name and unique ID. The Voter selects the election in which they want to participate. Election IDs should be published on some third trusted channel, so that voters could verify that it is indeed correct election.

When a specific election is chosen, the VoteApp verifies the signature of election data. If the signature does not verify, then the election data can not be trusted and the VoteApp stops the ballot casting process and displays an error to the voter. If the signature verifies, the VoteApp shows candidate list and gives the possibility to mark choice.

Marking Choices

The Voter sees the provided list of candidates and chooses one specific candidate for whom they want to vote. This reflects requirement *ballot-structure-simple*.

Complete Ballot Encryption

The VoteApp receives the Voter's intention to vote for a particular candidate and performs the CCE encryption of the vote and generates vote commitments and commitments for the well-formedness proofs. The encryption is performed with the help of VoteCollector, where the role of VoteCollector is the rerandomization of commitments, which are going to be published on the Bulletin Board for further verifications. The reason for rerandomization is to introduce receipt-freeness, so that later the Voter could not reproduce the created vote and show how they voted, thus helping to eliminate vote coercion.

When the rerandomization is made, rerandomized commitments and rerandomization proofs are sent back to the VoteApp where the VoteApp verifies that the performed rerandomization did not change Voter intention. If the rerandomization performed honestly, the rerandomized commitments are then signed by Voter private key and are published to the Bulletin Board.

The VoteCollector verifies signed commitments and finalized well-formedness proofs. If everything verifies, it adds the vote to the ballot box and sends rerandomized commitments, well formedness proofs, voter signature, and voter certificate to the Bulletin Board.

The Bulletin Board on receiving the data verifies well-formedness proofs of the encryption and the commitments signature with the Voter certificate. Additionally, the Bulletin Board performs verification of Voter eligibility by checking that the Voter certificate was issued from the root certificate authority defined in the election configuration and that that voter exists in the voter list and does not exist in the revocation list. If everything is correct, the public record is saved to the Bulletin Board, where any observer can make the same verifications.

The VoteCollector receives the success message from the Bulletin Board that indeed the public record is valid. It then signs the vote and adds it to the ballot box.

When the ballot casting process for the particular ballot is finished, the VoteApp receives success from the VoteCollector and shows the corresponding message to the voter.

Verifying the Public Record

Verification of the public record can be made by anyone with help of VoteApp. Any observer can select a particular election they want to verify. When an election is selected, VoteApp queries the audit data from the Bulletin Board for this election via the VoteCollector. The audit data consists of:

- vote public parts:
 - rerandomized commitments;
 - commitments signature;
 - well-formedness proofs;
 - voter certificate;
- election configuration;
- election configuration signature;
- voter list;
- voter list signature;
- revocation list;
- revocation list signature;
- Election Official certificate responsible for signature creation.

When the audit data is received, the observer can verify that the election configuration, voter list, and revocation list signatures are correct. Additionally, the observer can verify the public record by checking that the public record was made by an eligible voter and that it represents a valid vote commitment. The result of these verifications are shown to the observer.

Vote Submission Summary

Vote submission can be summarized in Figure 2.3, with a description of how different components interact with each other on vote submission.

D4.2 – Report on Architecture for Privacy-Preserving Applications on Ledgers

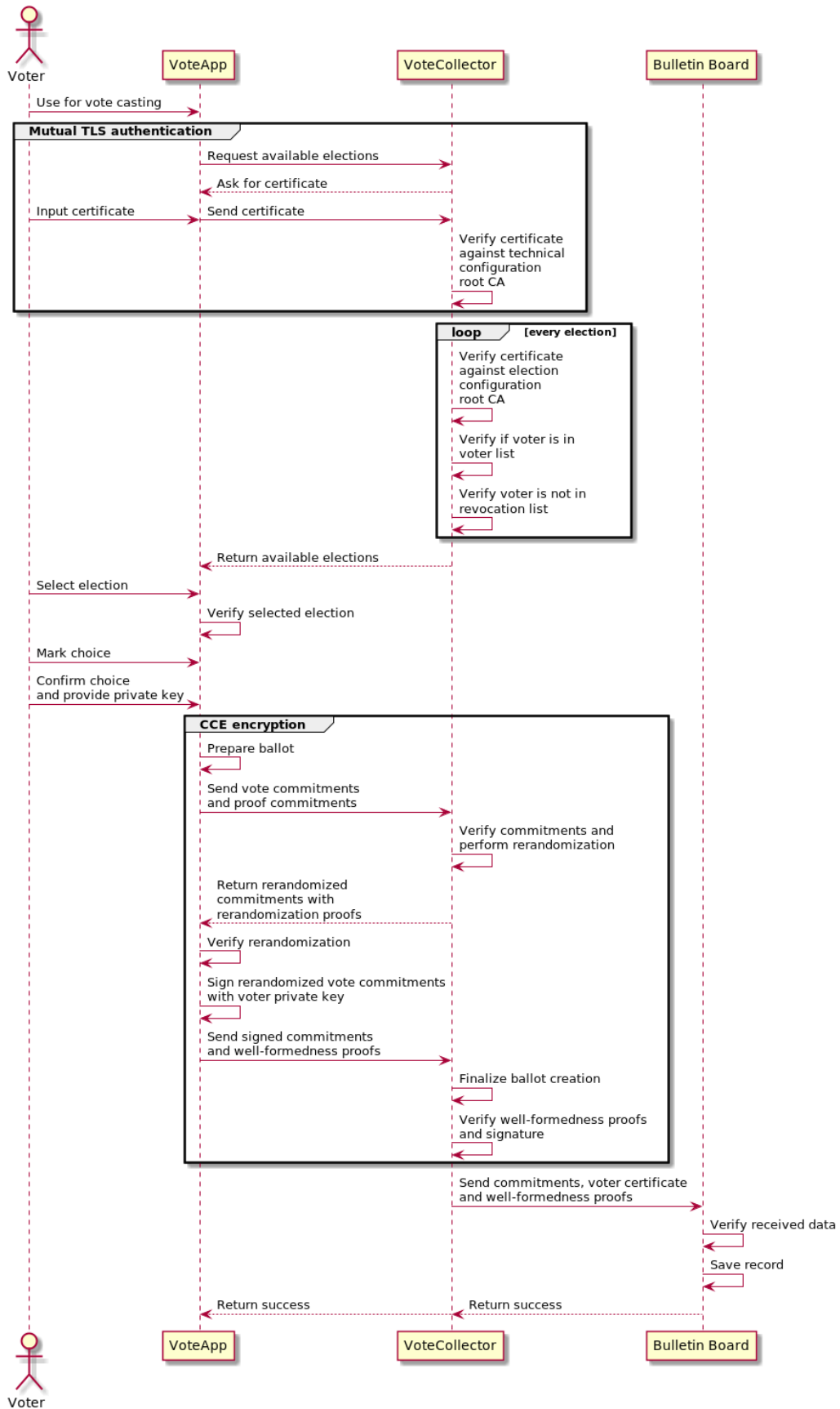


Figure 2.3: Vote submission summary.

2.5.3 Stage: Tally

The tally stage starts right after the election is past end time defined in the election configuration.

Exporting the Ballot Box

To export the ballot box, the Election Official can operate in two ways:

1. Physically accessing the VoteCollector.
2. By using ElectionManager which will receive the ballot box from the VoteCollector.

Ballot Box Processing

Ballot box processing is done by the Key Application. The ballot box processing consists of three steps.

1. Ballots are verified for integrity; this verification can be described by the following steps:
 - (a) Checking that every ballot contains all the information. This ensures that every ballot is valid and it is possible to verify the authenticity and integrity of the ballot. The completeness of every ballot is verified by checking that every entry in the ballot box is a tuple of the encrypted ballot, the ballot signature, the VoteCollector certificate and the election identifier. It also checks that no value in the tuple is empty.
 - (b) Checking that the signature on every ballot verifies. This ensures the integrity and authenticity of every ballot. This check is done by iterating over all entries in the ballot box and verifying the signature on the ballot using the VoteCollector certificate.

If integrity verification of some ballot fails, it is marked as an invalid and will not be sent to the next processing steps.

2. Ballots for which the voter has later cast another ballot by revoting are removed.
3. The ballots cast by voters who are included in the revocation list are revoked. The revocation list is used for example when the voter also casts a paper ballot in the polling station, in which case the online ballot is removed. This ensures that every voter can only vote once.

The Key Application verifies that the revocation list is signed by an authorized user and that the signature on the revocation list is correct, to ensure the integrity and authenticity of the revocation list. Then, every entry in the third stage ballot box is compared against the revocation list. If the voter identifier is included in the revocation list, then the entry is excluded from being added into the fourth stage ballot box. Otherwise, the whole entry is added to the fourth stage ballot box.

4. The ballot box is homomorphically aggregated so that CCE encryption of a voting result and corresponding commitment is achieved. This ensures the privacy of the voter as no vote is sent into decryption alone. The aggregated commitment of the ciphertext of a voting result is published to the bulletin board.

Results Decryption

The Key Application decrypts the encrypted results with the associated proofs, thereby opening a correct decryption, and stores the decrypted results for later publication.

Tally Phase Summary

A summary of the tally phase can be seen in Figure 2.3.

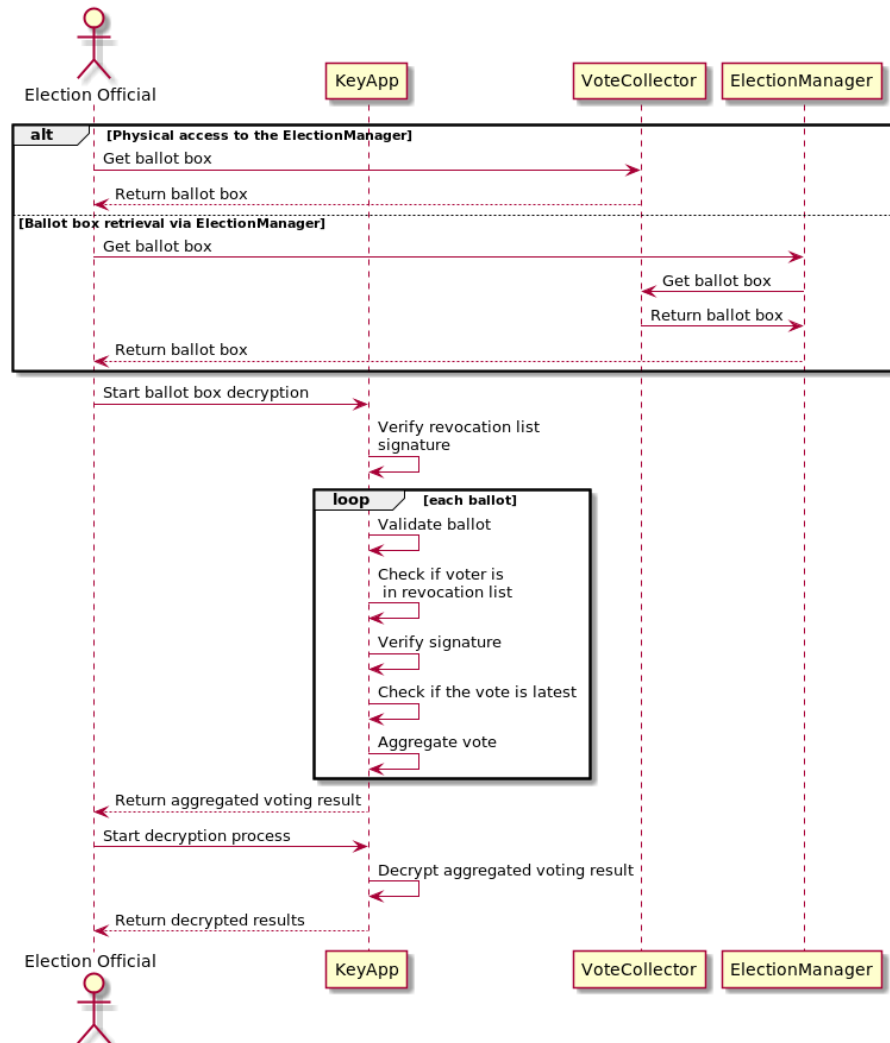


Figure 2.4: Tally phase summary.

2.5.4 Stage: Audit

As the Bulletin Board is distributed among the Election Official and the Auditor, the system is constantly being audited, because any modification on the Bulletin Board must be accepted by both parties.

Results publication

As results are published by the Election Official to the Bulletin Board via the ElectionManager, different steps are made to ensure that the election results were tallied correctly.

Firstly, the Bulletin Board performs validation of the published results by checking that results (counted number) for each candidate is given and that there exists an opening / proof for each candidate.

Secondly, Bulletin Board aggregates the published vote commitments in the voting phase and verifies their well-formedness proofs. If the revocation list was provided by the Election Official then the same logic will be

executed as in the ballot box processing with the Key Application, meaning that vote commitments published by the voters in the revocation list will not be aggregated.

Thirdly, it verifies the aggregated commitment for each candidate with the result (counted number) and the opening.

If all verifies, then this means that election results are indeed valid and trustworthy. The results are added to the Bulletin Board and are available for anyone, where any observer can make the same tally verification with data from the Bulletin Board in the VoteApp to ensure that ended the tally process is made correctly.

Audit Phase Summary

Audit phase summary can be seen in Figure 2.3.

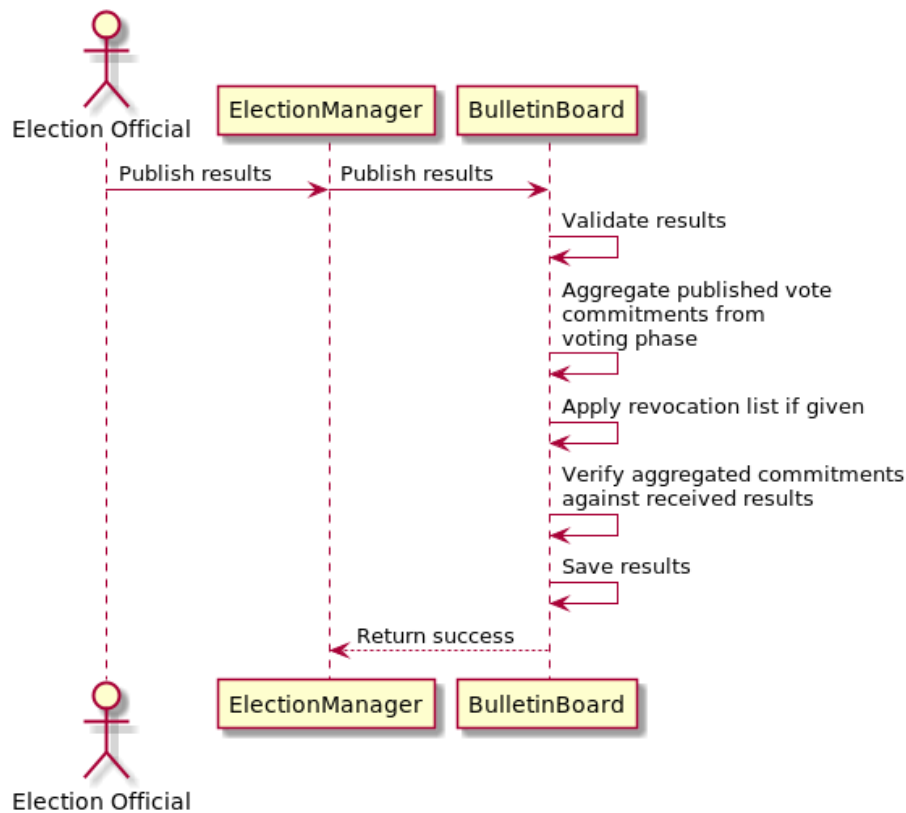


Figure 2.5: Audit phase summary.

2.6 Architectural Mechanisms

The following architectural mechanisms are used to compose the proposed system:

- unconditionally hiding, computationally binding Pedersen commitments;
- homomorphic public key encryption, lifted ElGamal;
- elliptic curve cryptography;
- pairings;
- commitment consistent encryption;

D4.2 – Report on Architecture for Privacy-Preserving Applications on Ledgers

- ciphertext/commitment re-randomization;
- non-interactive zero-knowledge proofs, Fiat-Shamir heuristics;
- hash functions;
- hash chain;
- Byzantine fault-tolerant consensus;
- PKI, digital signatures;
- anonymous credentials;
- TLS;
- threshold key generation.

The following technologies shall be put into use to implement aforementioned architectural mechanisms:

- JavaScript;
- Go;
- Python 3;
- HyperLedger Fabric;
- MIRACL Core Cryptographic Library;
- Docker;
- OpenSSL.

Chapter 3

UC2: Distributed Ledger for Health Insurance

3.1 Introduction

A health insurance scheme can be roughly modeled as interactions between three kinds of parties: *patients*, *insurers*, and *care providers*. Patients pay the insurers fixed monthly or yearly premiums based on their risk assessments and in exchange can get services from care providers paid for by the insurers.

Currently, the prevalent type of contract between an insurer and a care provider is “fee for service”, where the provider gets a pre-agreed fee for each treatment performed on insured patients. The drawback of this model is that it incentivizes the provider to prefer treatments with higher profit margins and sometimes even perform treatments not indicated for the patient’s condition.

At the other end of spectrum, in particular in primary care and nursing sectors, the “capitated” payment model is used, where the provider gets a fixed income stream per patient under its care, irrespective of how much treatment the patient really needs or receives. The downside of this model is that the care providers are incentivized to avoid performing treatments.

To motivate the providers to focus on best results for patients while simultaneously keeping costs under control, insurers are increasingly interested in outcome-based contracting, also called “pay for performance”. In this model, the payments that providers receive depend on achieving certain measurable outcomes, such as reducing the mean time off work for a certain diagnosis or lowering the number of patients with blood pressure exceeding a given threshold.

To be able to affect such broad outcomes, care providers with multiple different specializations typically join into an “accountable care organization” (ACO) to share the responsibilities as well as the financial rewards and risks. In practice, most ACO contracts are not pure outcome-based, but combine different payment models in various proportions.

3.2 Requirements

- **Consistency:** The primary goal of using distributed ledger technology is to ensure that all parties have the same view of the records pertaining to them.
- **Privacy:** As medical records require long-term privacy, they should not be posted on a ledger, even in encrypted form. Instead, we propose to use the ledger for sharing only cryptographic commitments of the records and exchange actual data via off-ledger point-to-point channels.
- **Integrity of medical histories:** The commitments posted to the ledger should enable verification that each update only adds new events to a patient’s medical history, but does not change or delete any existing events.

- **Consistency of reports:** The commitments posted to the ledger should also enable verification that the reported values of the metrics are indeed computed from the latest state of the patients’ records.
- **Accountability:** When records are passed from one care provider to another, both their provenance and patient consent should be tracked to enable evidence-based accountability of care for the patients’ health as well as the privacy of their medical histories.

3.3 High-Level Architecture

Both coordination of work within an ACO and reporting to the insurer on the objectives require sharing of data. While it is usually easy to get consent to share the medical records when a patient is referred to another provider in the ACO, it is not the case when reporting to the insurer is concerned, as patients (justifiably) fear that sharing detailed medical records might cause discrimination in future insurance offerings.

Also, a provider may participate in multiple contracts with different insurers. Thus, even providers within the same ACO may not want (or even be allowed) to share all of their records (neither with any one insurer nor with other members of the ACO). Therefore, we foresee the ACO setting up a data exchange to facilitate on-demand point-to-point transfers of specific records, and a ledger-based commitment mechanism to help ensure integrity and consistency of those records.

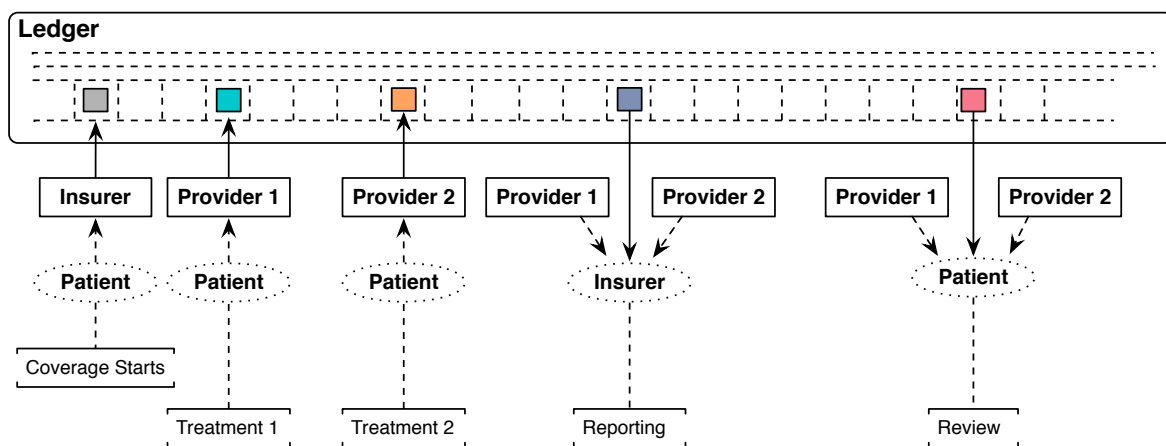


Figure 3.1: Healthcare provisioning process.

The overall process of providing health-care to a patient is as follows (see Figure 3.1):

- When a patient signs up for coverage with the insurance provider, the insurer updates the list of covered patients and posts a commitment of the new state to the ledger.
- When a patient visits a care provider, the provider checks that the patient has coverage, performs the necessary analyses and / or treatments, records the details of the visit in their own records management system, and posts a commitment of the new state to the ledger.
- When the patient visits another care provider (whether on their own or by referral), the receiving provider obtains patient consent, then retrieves the relevant records via a point-to-point channel and checks that they match the commitment posted to the ledger by the sending provider, then performs the necessary services, records the details of the new visit in their own records management system, and again posts a commitment of the new state to the ledger (see Figure 3.2).

- At the end of each accounting period, the providers compute the performance indicators agreed to in the payment model (which could include both “fee for service” and “pay for performance” type metrics) and submit the results to the insurer who uses the commitments on the ledger to verify the reports are consistent with the underlying records before issuing payments (see Figure 3.3).
- Optionally, patients may also review their own records. While we do not expect all patients to use this option, even a small percentage of them doing so can act as a deterrent against care providers falsifying data.

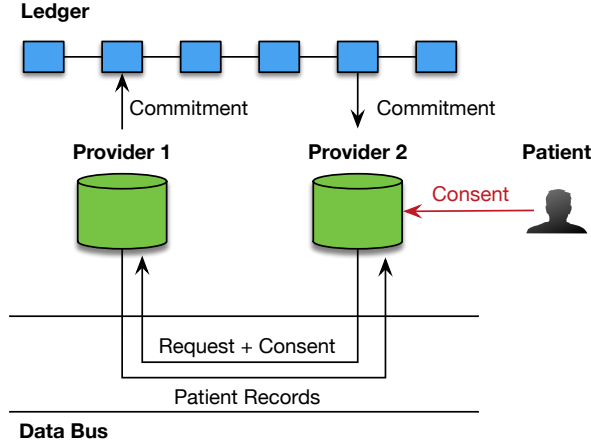


Figure 3.2: Referral data flow.

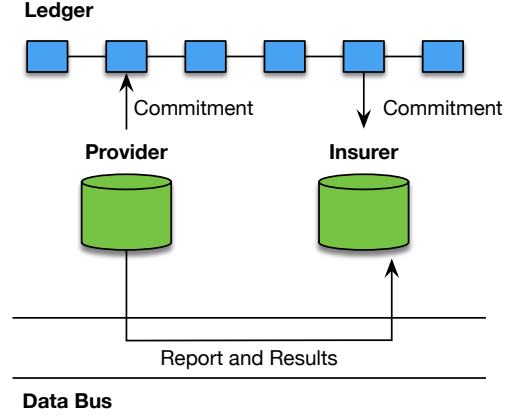


Figure 3.3: Reporting data flow.

3.4 Data Model

For the purposes of ledger interactions, each patient’s record will be modeled as a stream of events. Each event in turn is a tuple consisting of the patient ID, an event type, and zero or more attributes.

The number and order of attributes will be predefined for each event type. In the first prototype only integer-valued attributes will be supported (when other types are used on business level, they will be encoded as integers for the ledger interactions).

In the initial prototype, each patient’s record will be managed by their primary care provider and when the patient interacts with any other party, the corresponding events are sent to the primary care provider for inclusion in the patient’s record.

3.5 Component Interactions

3.5.1 Updating Patient Records

After new events have been added to patients’ records, the care provider c maintaining the records will post an updated commitment to the ledger. The update will be posted in two steps:

- First, a commitment $C(\Delta_{c,t})$ of the added events will be posted. This commitment will have no cryptographic proof attached to it, as the factual correctness of the underlying data can in any case be verified only by an audit.
- Second, a new global commitment $C(\Sigma_{c,t})$ will be posted, along with a proof that $C(\Sigma_{c,t})$ corresponds to the result of adding the events under the commitment $C(\Delta_{c,t})$ to the previous global state under the commitment $C(\Sigma_{c,t-1})$.

Such two-step update process enables verification that each update only adds new events, but does not delete or change existing ones.

Note that the updates are posted as globally aggregated batches, which is both more efficient and also more privacy-preserving than posting multiple smaller per-patient updates.

3.5.2 Reviewing Patient Records

A patient reviewing their own records or a specialist receiving a referred patient's records from the primary care provider c will verify it against the latest posted global commitment $C(\Sigma_{c,t})$.

The subset Π_p for the patient p comes with a proof that it is exactly the subset of events from the global set $\Sigma_{c,t}$ under the commitment $C(\Sigma_{c,t})$ that satisfies the condition that the first element of each event vector has the value p .

3.5.3 Compilation of Reports

Members of an ACO will use a secure multi-party computation (MPC) protocol to compile a report on the cohort of patients, including the aggregate values agreed to in the care provisioning contract along with a zero-knowledge proof of consistency of the underlying data with the commitments on the ledger. The compilation of the report will consist of the following steps:

- First, each care provider c will extract one or more subsets of events $S_{c,i}$, either from their set of records $\Sigma_{c,t}$ or from a previously extracted intermediate $S_{c,i'}$, each corresponding to some sub-condition T_i of the form “the events such that the k -th coordinate the event vector has the value between v_1 and v_2 ”, where the specific conditions are chosen so that the performance metric to be reported can be expressed as a combination of those. For each of those subsets, a commitment $C_{c,i} = C(S_{c,i})$ is computed and a proof is given that the subset under the commitment $C_{c,i}$ is indeed the subset of events satisfying the condition.
- Second, all care providers will feed their inputs $S_{c,i}$ and the corresponding proofs into an MPC process that will combine the inputs and compute the requested aggregate function, along with a proof that the computed aggregate value is correct.

An insurer receiving the report can verify the proofs supplied with the report against the commitments on the ledger or the intermediate commitments also delivered with the proof, as appropriate, before accepting the aggregate value in the report as the basis for payment.

3.6 Cryptographic Tools

With the data model outlined in Section 3.4, the interactions described in Section 3.5 can be supported with a commitment scheme and an MPC protocol with the following mutual properties.

3.6.1 Commitment Scheme

The commitment scheme needs to allow computing commitments on sets of vectors with integer coordinates with the following three additional requirements:

- Given two sets S_1 and S_2 and their union $S = S_1 \cup S_2$ with the corresponding commitments $C_1 = C(S_1)$, $C_2 = C(S_2)$, and $C = C(S)$, it must be possible to give a proof that the set under the commitment C is indeed the union of the sets under the commitments C_1 and C_2 .
- Given a base set S and a subset S' consisting of the members of S satisfying the condition that the value of the k -th coordinate is between the given values v_1 and v_2 , and the corresponding commitments $C = C(S)$

and $C' = C(S)$, it must be possible to give a proof that the sets under the two commitments indeed have the required relationship:

$$S' = \{e = (e_1, e_2, \dots, e_m) : e \in S \text{ and } v_i \leq e_k \leq v_2\} .$$

- Given a set S , the corresponding commitment $C = C(S)$, and an aggregate value A computed over the values of the k -th coordinates of the vectors in S , it must be possible to give a proof that A is indeed computed in this way from the elements of the set under the commitment C . We foresee that the ability to compute sums and counts as the aggregates will be sufficient at least for the initial prototype.

We expect that a derivative of the Pedersen commitment scheme will be able to fulfill these requirements.

3.6.2 Multi-Party Computation Protocol

The MPC protocol must have the ability to compute two kinds of results on the sets and their corresponding commitments:

- Given n sets $S_{c,i}$ and their commitments $C_{c,i} = C(S_{c,i})$, the MPC must be able to compute the union

$$S_i = S_{1,i} \cup S_{2,i} \cup \dots \cup S_{n,i}$$

and the corresponding commitment $C_i = C(S_i)$, along with a proof that the set under the commitment C_i is indeed the union of the sets under the commitments $C_{c,i}$.

- Given a set S and the corresponding commitment $C = C(S)$, the MPC must be able to compute an aggregate value A (the sum or count) over the values of the k -th coordinates of the vectors in S , along with a proof that A is indeed the given aggregate over the elements of the set under the commitment C .

Chapter 4

UC3: University Diploma Record Ledger—The DIPLOMATA System

4.1 Introduction

DIPLOMATA is a system that enables the verification of authenticity for academic degrees in a privacy-preserving manner. It employs cryptographic techniques in order to protect privacy and records transactions in a distributed ledger to hold all involved parties accountable.

Three distinct entities are involved: ISSUERS, HOLDERS, and VERIFIERS. An ISSUER is an organization (e.g., a university) that issues qualifications. HOLDERS are persons that have qualifications (i.e., graduates) and may wish to present them to an interested third party. This third party can be an organization or a person that wishes to verify a qualification, i.e., a VERIFIER. To do so, the VERIFIER relies on evidence provided by the qualifications's ISSUER. The aforementioned concepts are in line with the World Wide Web Consortium's Verifiable Credentials Data Model [W3C19]. Moreover, an external *Auditor* may inspect the process.

DIPLOMATA employs a blockchain to record all transactions. Only the three aforementioned entities can write to the distributed ledger. The ledger is accompanied by an application that is responsible for key management, messaging, and anything else not covered by the cryptographic protocols per se.

4.2 Requirements

In the following, we specify a number of basic security and privacy requirements that we have taken into account during the development of DIPLOMATA.

- **Title Privacy:** Titles are private data, meaning that they should not be accessible to anyone else except for the ISSUER.
- **Access Control:** The data inside the titles are personal data belonging to their HOLDERS. The HOLDERS should be able to control who and when gains access to their titles. In addition, HOLDERS should control which parts of the verifiable credentials can be presented to a VERIFIER (e.g. full transcript or bare title).
- **Accountability:** All entities should be held accountable for all their actions. For instance, ISSUERS are not expected to issue forged titles. Furthermore, VERIFIERS should accept only valid credentials, and be held responsible for any proof they leak.
- **Auditability:** An honest external auditor should be able to inspect all the recorded transactions.

In addition to the above, we adopt the following assumptions:

- A HOLDER should be able to release any information they choose to any VERIFIER they choose. The system should protect against a corrupt VERIFIER extracting additional information.
- DIPLOMATA will not be able to tell if a qualification is forged or not, but it should guarantee that no forged qualification can be both acceptable and hidden.
- Proofs from the ISSUERS to the VERIFIERS should not be leaked, but DIPLOMATA cannot prevent proof leaks. However, DIPLOMATA should guarantee that if a leak happens, the VERIFIER must always be held accountable. There will not be anonymous leaks.
- An auditor is special because they can request opening of arbitrary commitments.

4.3 Approach

To allow qualification HOLDERS to certify that they indeed have acquired a qualification from an institution, DIPLOMATA involves a cryptographic *protocol* and an *application* to support it. First, we describe all protocol steps in an abstract way and discuss the basic workflow. Then, we present the DIPLOMATA protocol in detail, highlighting its cryptographic properties.

4.3.1 Overview

An ISSUER awards a qualification to a specific HOLDER. A commitment to the qualification document, including a reference to the HOLDER's identity, is recorded into an immutable ledger (1). The ISSUER privately sends the Holder a reference to the recorded entry.

The HOLDER can request that the ISSUER certify the qualification to a particular VERIFIER. The request indicates both the qualifications's commitment and the VERIFIER; it is recorded into the immutable ledger (2). The HOLDER privately sends the VERIFIER the qualification document that will be certified by the ISSUER.

The ISSUER validates the HOLDER's request and creates a certificate addressed to the VERIFIER. The certificate convinces anyone who has access to the commitments that the document certified here has been previously committed to at step (1).

The certificate convinces the VERIFIER that the qualification document received from the HOLDER is considered valid and current by the ISSUER. The certificate is recorded by the ISSUER into the immutable ledger (3). Neither the HOLDER nor the VERIFIER nor anyone else can create an (undetectably) legitimate certificate instead of the ISSUER.

The VERIFIER creates an acknowledgment that they have received and verified the certificate. There are three cases: 1) the certificate is malformed, 2) it is well-formed but does not match the qualification document received from the HOLDER, or 3) it is well-formed and matches the document. The acknowledgment is recorded by the VERIFIER into the immutable ledger (4).

4.3.2 The DIPLOMATA Protocol

Preliminaries

- We consider that an ISSUER I makes an AWARD for QUALIFICATION t to a HOLDER L .
- The HOLDER may request that the ISSUER provide proof of the AWARD to a designated VERIFIER V .
- We use the ElGamal encryption system. Any actor A is identified by their public key, g^A , assuming generator g and private key A . The ElGamal encryption of a message m to actor A is the pair $(g^r, g^{Ar}m)$, where r is a random number. Although the public key of an actor identifies the actor, it is not necessary that it is linked to a publicly known identity; actors could agree on an identity scheme like Decentralized Identifiers (DIDs) [W3C20].

- Given a generator g and a public key g^P of a prover P , a non-interactive (Fiat-Shamir) proof of a DDH tuple $\langle g^P, g^a, g^{Pa} \rangle$ for a value a using the Chaum-Pedersen protocol for $\langle g, g^P, g^a, (g^P)^a \rangle$ is written as $\text{NIDDH}_P(a)$.
- Given a generator g and a public key g^P of a prover P , a non-interactive (Fiat-Shamir) proof of re-encryption of ciphertext c to c' using the Chaum-Pedersen protocol $\langle g, g^P, c, c' \rangle$ is written as $\text{NIRENC}_P(c, c')$.
- A cryptographically secure hash of content c is written as $\mathcal{H}(c)$.
- To privately communicate a message m of arbitrary length to an entity A over the ledger, we assume an encryption scheme wherein entities are identified by their ElGamal keys. We write $\mathcal{E}_A(B)$. In practice, a hybrid encryption scheme can be used instead for performance.
- Each ledger entry is a tuple consisting of a tag and additional content. The possible tags are AWARD, REQUEST, PROOF, ACK, NACK, and FAIL. Entries are signed by the entity that publishes them. We assume an appropriate signature scheme. Every entry acquires a unique serial identifier s upon publication. We write $s \leftarrow^A \text{entry}$ to indicate that A has signed and published the *entry* identified by s .

Protocol Steps

1. An ISSUER I publishes an AWARD for a QUALIFICATION t of HOLDER L as:

$$c = (c_1, c_2) = (g^r, g^{Ir} \mathcal{H}(t))$$

$$s_{\text{awd}} \leftarrow^I \text{AWARD}, c$$

❏ The ISSUER commits to an encrypted hash of the qualification document t and sends s_{awd} to HOLDER L . The ISSUER remembers the randomness r used for encryption.

2. The HOLDER L publishes a request for proof of s_{awd} addressed to a VERIFIER V as:

$$s_{\text{req}} \leftarrow^L \text{REQUEST}, s_{\text{awd}}, g^V$$

❏ The request signature can be verified by the ISSUER in order to identify the HOLDER and ensure that this is the true holder of the qualification committed to at s_{awd} . The VERIFIER is specified with their public key g^V .

3. The ISSUER proves the AWARD to the VERIFIER:

$$c' = (c'_1, c'_2) = (g^{r'} c_1, g^{Ir'} c_2) = (g^{(r+r')}, g^{I(r+r')} \mathcal{H}(t)) = (g^{\tilde{r}}, g^{I\tilde{r}} \mathcal{H}(t))$$

$$s_{\text{prf}} \leftarrow^I \text{PROOF}, s_{\text{req}}, c', \text{NIRENC}_I(c, c'), \mathcal{E}_V(g^{I\tilde{r}}, \text{NIDDH}_I(\tilde{r}))$$

❏ The ISSUER publishes a re-encryption of the commitment using a fresh randomness $\tilde{r} = r + r'$ and a non-interactive Chaum-Pedersen proof of the re-encryption. Encrypted for the VERIFIER they also publish the decryptor for the re-encrypted commitment as well as a non-interactive DDH proof of decryption.

4. The VERIFIER publishes an acknowledgement:

$$s_{\text{ack}} \leftarrow^V \text{ACK}, s_{\text{prf}}, \mathcal{E}_I(g^{I\tilde{r}})$$

$$s_{\text{ack}} \leftarrow^V \text{NACK}, s_{\text{prf}}, \mathcal{E}_I(g^{I\tilde{r}})$$

$$s_{\text{ack}} \leftarrow^V \text{FAIL}, s_{\text{prf}}$$

¶ The VERIFIER validates the provided proofs and checks that the qualification document at hand hashes to $\mathcal{H}(t)$. Using an external authentication mechanism they validate that the signer of the original commitment and of the proof is indeed the ISSUER of the qualification document. They acknowledge having the means (i.e., the decryptor $g^{I\tilde{r}}$) with which to recover the correct cleartext.

Addressing Possible Attacks

1. *Readers access qualifications:* Qualification t or $\mathcal{H}(t)$ is never published in cleartext nor can it be retrieved from the ciphertexts.
2. *Proof without commitment:* The proof is based on a re-encryption of the award commitment. The correctness of the re-encryption is openly verifiable (proof $\text{NIRENC}_I(c, c')$ at step 3). The VERIFIER is convinced by the NIDDH proof of decryption that the ISSUER knows \tilde{r} and thus the decryptor $g^{I\tilde{r}}$ is not fabricated.
3. *Proof of false commitment:* The ISSUER can issue a false commitment. An auditor can track a suspicious proof back to the commitment and force the ISSUER to reveal it.
4. *Proof of a different qualification:* The HOLDER can check that the proof is based on a re-encryption of the commitment at s_{awd} . The HOLDER must trust the ISSUER that s_{awd} contains indeed a commitment for their own QUALIFICATION.
5. *Verifier leaks proof:* The VERIFIER cannot convince a third party of the commitment they have received unless they also reveal the related NIDDH decryption proof, which they cannot alter or create from scratch. Since this proof has been recorded as designated to the VERIFIER, any convincing leak is traceable back to them.
6. *False acknowledgment:* The ISSUER or an auditor with access to the ISSUER's key can verify the acknowledgment.

4.4 Architectural Considerations

The DIPLOMATA protocol is an essential component of the DIPLOMATA system, but it is only a part of it. To create a working system in the context of PRIViLEDGE, additional components must be developed, or sourced from existing libraries, and glued together. Essential architectural components include (see Figure 4.1):

- **Identify and Key Management System:** As actors are identified by their public keys, a mechanism must exist for linking actors to their keys and certifying that a key belongs to a certain actor. A Public Key Infrastructure (PKI) could be used for that purpose; alternatively, as already mentioned, a DID could be used if it is desirable not to rely on a centralised public key mechanism.
- **Distributed Ledger Abstraction Layer:** The DIPLOMATA protocol does not rely on a particular blockchain implementation. To maintain this aspect of implementation neutrality, access to the underlying ledger should be mediated by an intermediate, blockchain-abstraction layer. Although it is likely that a digital credentials application would run on a permissioned ledger (with nodes maintained by the institutions issuing the degrees), such an abstraction layer would allow it to run on public ledgers as well.
- **Basic Crypto Layer:** DIPLOMATA makes use of several cryptographic operations, such as hashes, signatures, and cryptographic schemes; it also uses schemes based on cryptographic primitives, such as Chaum-Pedersen proofs. These will be bundled in a single crypto layer that will be used by the other components of DIPLOMATA. Bundling all necessary cryptographic operations and schemes in a separate layer separates the cryptographic implementation from their use; thus, cryptographic operations will be available either as library calls, or as API calls (e.g., through JSON), depending on the implementation choices (such as programming languages) for the other DIPLOMATA components.

- **Transaction Logic Layer:** The various steps required by DIPLOMATA will be implemented in a separate layer, which will use the aforementioned three components. This will ensure that the transaction logic will be at a level close to the original protocol description, instead of getting tangled in lower-level details, or being extended to higher, user-facing functions.
- **Presentation Layer:** The DIPLOMATA actors, i.e., ISSUER, HOLDER, VERIFIER, will interface with this layer, which will include the necessary user interfaces. These will be implemented using a RESTful architecture, so that different user interfaces (web-based or mobile) will use the same calls to cater for different device affordances.

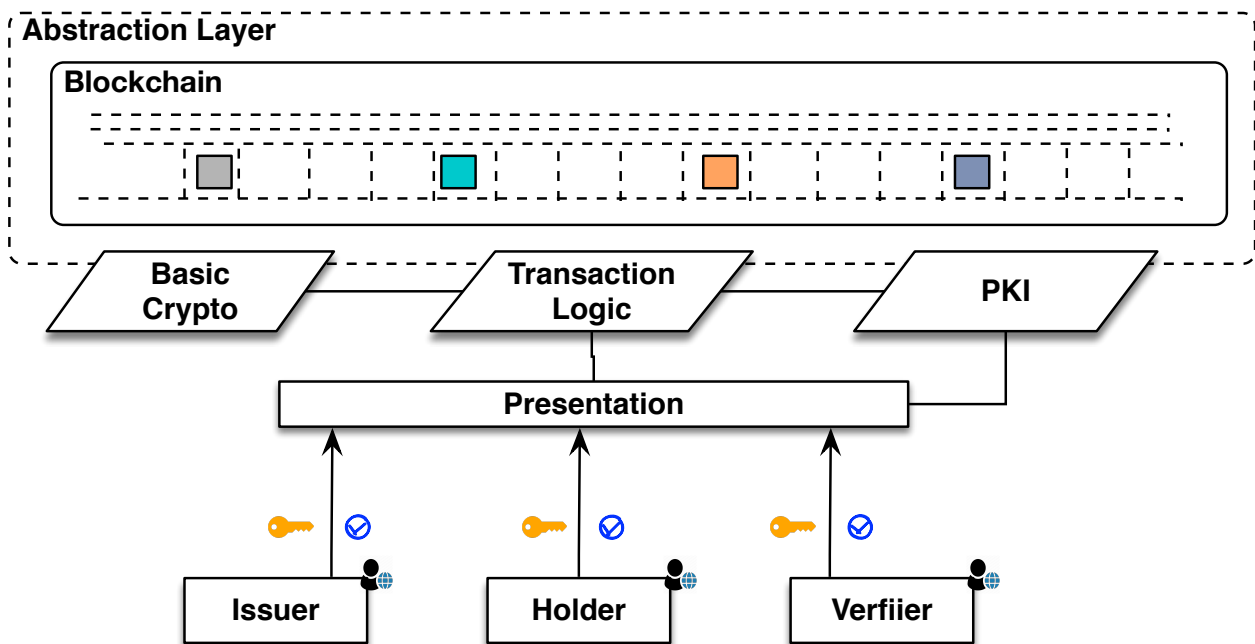


Figure 4.1: DIPLOMATA Architecture.

Chapter 5

Toolkit for Zero-Knowledge Proofs in Ledgers

5.1 SNARK Setups

Since their deployment as part of Zcash, zk-SNARK systems have found widespread use as a mechanism for improving the privacy of ledger transactions. In Zcash, SNARKs enable users to make anonymous transfers between accounts, without substantially increasing the size or verification time of transactions. Their succinctness comes however at a price, the party that generates the reference string for the SNARK system is given an inordinate amount of power. It can for instance print its own money, circumventing the ledger rules, or could potentially even subvert privacy. A potential solution (adopted in the roll-out of Zcash) can be a public ceremony, where the reference string is generated by distributed partners, and then destroyed, while the whole process is broadcast and recorded. Although a public ceremony has publicity benefits, they are expensive, unwieldy, and at the end one has to trust that the ceremony is a genuine one and not a piece of theatre. As a toolkit for zero-knowledge proofs in ledgers, PRIViLEDGE will develop tools that allow to harness the benefits of SNARKs without the downsides of a trusted setup.

5.2 Subversion-Resistant Zero-Knowledge

UT have implemented a subversion-resistant zk-SNARK [Abd+17] in the Libsnark library; that is the most well-known library for zk-SNARKs, it is developed in the SCIPR Lab and released under the MIT License. This is the first implementation of a zk-SNARK where the prover does not need to trust the CRS generators and instead it checks the well-formedness of CRS elements with an efficient CRS Verification (CV) algorithm. After presenting the subversion-resistant zk-SNARK in [Abd+17], there have been some other subversion-resistant zk-SNARKs but none of them has been implemented in a manner that would allow for their use in practical scenarios.

Following the proposed scheme, which is a variation of Groth's zk-SNARK [Gro16], UT's implementation is a modified version of the implementation of Groth's zk-SNARK available in the Libsnark library ¹. The main difference between the subversion-resistant zk-SNARK proposed in [Abd+17] and the original scheme proposed in [Gro16], is that the former proposes an efficient CV algorithm that should be executed by the prover to eliminate the trust on the CRS generators. Abdolmaleki et al. [Abd+17] have presented two different types of CV algorithms for verifying the well-formedness of CRS elements in Groth's zk-SNARK. They showed that their CV with batching technique is considerably more efficient than the standard CV algorithm. This toolkit implements both CV algorithms, indexed by standard or batched, for Groth's zk-SNARK [Gro16]. Recall that generally a non-subversion-resistant zk-SNARK has three algorithms known as *CRS generator*, *Prover* and *Verifier*. The CRS generator takes a security parameter and generates a CRS that later will be used for proof generation and proof verification. The algorithm Prover, given the CRS, a statement and the corresponding witness, generates a proof. Finally the algorithm Verifier, given the CRS, the statement and a proof, verifies if

¹Available on https://bitbucket.org/Bagheri/subversion_qap_implementation/src/master/

the proof is valid or not.

In the implementation of subversion-resistant zk-SNARK [Abd+17], the command

```
$src/zk_proof_systems/ppzksnark/r1cs_gg_sub_ppzksnark/profiling/...  
profile_r1cs_gg_sub_ppzksnark 1000 10 Fr batched
```

executes the subversion-resistant zk-SNARK (the algorithms CRS generator, CRS Verifier, Prover, Verifier) on a Rank1 Constraint System (R1CS) instance with 1000 equations, an input consisting of 10 field elements and batched CV algorithm.

The command

```
$src/zk_proof_systems/ppzksnark/r1cs_gg_sub_ppzksnark/profiling/...  
profile_r1cs_gg_sub_ppzksnark 1000 10 Fr standard
```

executes the subversion-resistant zk-SNARK (the algorithms CRS generator, CRS Verifier, Prover, Verifier) on an R1CS instance with 1000 equations, an input consisting of 10 field elements and standard CV algorithm.

The PRIViLEDGE toolkit for zero-knowledge proofs in ledgers will ideally guarantee subversion soundness. This could be achieved by maintaining the above integration into the Libsnark library, or through the more ambitious path detailed next.

5.3 Soundness with Updateable Setup

Subversion-resistant zk-SNARK address the issue of privacy leaks in presence of a subverted setup. The setup process, however, still needs to be trusted with respect to the soundness of the proof system.

UEDIN have developed Sonic [Mal+19], a new zk-SNARK for general arithmetic circuit satisfiability. Sonic requires a trusted setup, but unlike conventional SNARKs the Structured Reference String (SRS)² supports all circuits (up to a given size bound) and is also updatable, so that it can be continually strengthened. This addresses many of the practical challenges and risks surrounding such setups. Sonic’s SRS is linear in size with respect to the size of supported circuits, as opposed to the scheme by Groth et al. [Gro+18], which scales quadratically. The SRS in Sonic also does not need to be specialized or pre-processed for a given circuit. This makes a large, distributed setup process a practical reality. In blockchain-related applications the integration between setting up a SRS and using the resulting SNARK can be realised almost seamlessly. As has been shown in [KKK20], it is possible to implement a protocol in such a way that it allows a procedure of sequential SRS updates in the setup phase, where users can contribute to the common randomness, and are also incentivised to do so, additionally to the off-protocol incentive to obtain a secure SRS. The protocol uses the honest stake majority assumption and the resulting ledger properties, especially chain quality, to decide when it is safe to finish the setup phase, which ultimately then fixes a specific SRS for further use. This appealing way to resolve the pre-processing trusted setup issue is what motivates the suggestion to use Sonic in blockchain-related applications, including private smart contracts.

5.4 Private Smart Contracts and zkay

Achieving privacy in smart contracts has attracted research effort recently [Kos+15; Kal+18; Bow+18; Bün+19; Che+19b]. Solutions that target this direction attack it from many different angles, varying by their use of cryptographic techniques (which implies diverse trust assumptions), and by the medium to which they are applied (the hardware, the blockchain, or the smart contract language itself). In this project we suggest to focus on another solution, called zkay [Ste+19]—a toolbox for writing smart contracts on Ethereum. Our choice is motivated by the fact that zkay provides a balanced focus on programming language aspects, its use of Non-Interactive Zero

²The “Structured Reference String” substitutes, and is preferred to, the term “Common Reference String”, see <https://docs.zkproof.org/pages/reference/versions/zkproof-implementation-20180801.pdf>

Knowledge proofs (NIZKs) is fairly straightforward, but nevertheless its expressivity is quite high. The main idea behind zkay is that users can define privately owned variables, which are represented as ciphertexts on the public smart contract, and updates to these variables can be issued using NIZKs, thus maintaining privacy. The zkay system consists of two parts: the zkay language, more precisely an extension of the Solidity language, and the compiler that transforms the smart contract code into two independent pieces of code. The first one is the Solidity contract that can be run, for example, on the Ethereum blockchain, and the second one is a circuit description of the code that is supposed to be run inside a zk-SNARK. The implementation of zkay is available on Github ³.

The language of zkay is different from the original Solidity language in having privacy annotations that specify the ownership of internal script variables. These privacy annotations allow the author of the private smart contract, for instance, to limit the access to the concrete script variable to a single address, so that the variable can be read exclusively by the owner of that address. There are a few other important language features, such as careful handling of statements for declassifying the secret information, that in conjunction with access annotations provide a user environment that is sufficient to implement various smart contracts with nontrivial privacy access structure. The main example in the paper, for instance, presents a smart contract that emulates a simplistic medical database. This contract stores a variable mapping each user to a boolean value that indicates whether this user is in the risk group or not, and a global counter of all the users in the risk group. The entity “hospital” can then assign these values and increase a private counter (that is not visible to anyone else), and users can read their own risk assignment from their own private variable (and again, nobody else can read from it). The smart contract guarantees that the counter is consistent with the risk information given to the users.

After a zkay script is written and passed to the compiler, it is transformed into a Solidity contract that uses NIZKs to perform the operation that is meant to be private. For example, in the hospital example, the update by the hospital would be represented by a transaction with a NIZK certifying that the new user risk values (in their variables) were computed correctly, with respect to the smart contract function that the hospital triggers, and its private inputs. The concrete NIZK expression is also created (extracted) from the original contract code in the zkay language, and compiled to the ZoKrates language. ZoKrates [ET18b], which is a toolkit for zk-SNARKs on Ethereum, provides a compiler that translates the original script into the SAP (Square Arithmetic Programs) language that is “native” to the NIZK used. The particular NIZK with which zkay parameterises the ZoKrates implementation⁴ is the zk-SNARK by Groth and Maller (GM17, [GM17]), which is implemented as part of the libsnark⁵ library used by ZoKrates. The ZoKrates language is a user-oriented imperative language, but the functionality it offers is still more low-level than the one of zkay. It can be thought of as a high-level abstraction of an arithmetic circuit—for example the only type it supports is a field element. The extraction (transformation) procedure of zkay is itself one of its main results, which includes the static analysis of the code.

The way zkay is used practically depends mostly on ZoKrates. The generated Solidity code can be used as any other contract, and this code includes the implementation of the SNARK verification itself, together with elliptic curve operations and concrete parameters. For a user to generate such proofs (in order to interact with the contract) they would need to use ZoKrates, either directly, or indirectly via underlying calls from their wallet. To our best knowledge, the latter solution hasn’t been yet implemented, although it is definitely technically possible. But, even though the former (direct) approach is not very user friendly, it nevertheless does not require any complicated technical skills. Since the proof size is small, it can be represented by a compactly encoded string (for example, using base64 encoding), which can be copied from the output of ZoKrates console call and pasted to the corresponding field in a wallet interface, as the user would do with any other contract input. The visual description of interactions between the components is presented in Figure 5.1. The following article explains technical aspects of using ZoKrates, including the ones that just have been mentioned: “Getting started with zkSNARKs Zokrates”⁶.

³<https://github.com/eth-sri/zkay>

⁴<https://github.com/zokrates/zokrates>

⁵<https://github.com/scipr-lab/libsnark>

⁶<https://blog.gnosis.pm/getting-started-with-zksnarks-zokrates-61e4f8e66bcc>

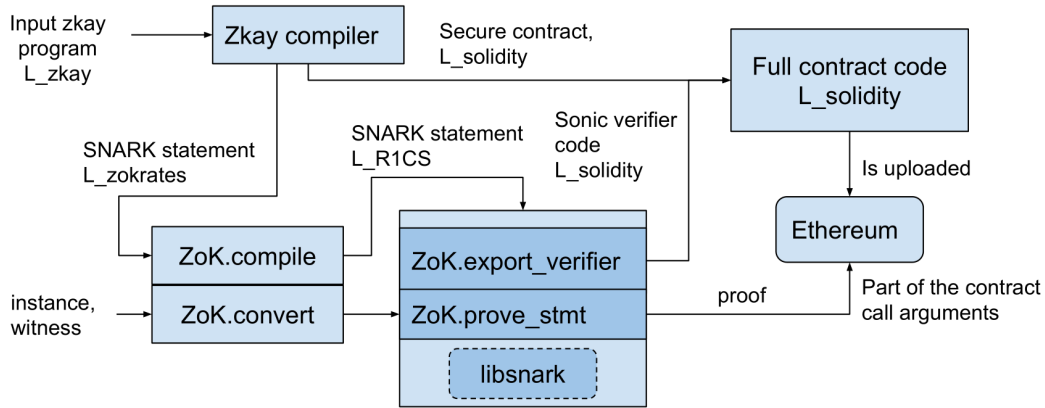


Figure 5.1: zkay’s compilation and execution process. The diagram is simplified and does not strictly reflect ZoKrates code dependencies. In reality, ZoKrates is a single multi-command executable, which here is presented as a set of two groups of separate functionalities. The compile/convert box is shown as one box, because the same machinery is used both to compile the circuit to R1CS and to compile the circuit inputs to R1CS inputs. Export and prove statement box is joined because they both depend on the compiled circuit, and are conceptually similar.

5.5 Practical Private Smart Contracts with Updatable SRS

The particular SNARK used in zkay, GM17, as it is implemented in libsnark, can accept two types of (circuit description) languages as input: R1CS and SAP. Practically speaking, both of them are intermediate stage representation languages. GM17 itself uses SAP internally, but libsnark provides a conversion module that transforms R1CS programs to SAP. Even though the newer eprint version of GM17 adds a CRS verification routine (that is used to prove subversion zero-knowledge), and GM17 is used in libsnark, it is currently not the case that any of the libsnark components support the updatable SRS paradigm, although no apparent limitation to do so exists.

While observing the technical possibility to join the efforts of practical private smart contract systems and research on trustless NIZKs, we propose to extend the functionality offered by zkay, by allowing users to choose the Sonic zk-SNARK instead of GM17 in a practical implementation. This section elaborates on different technical details of such a solution, and suggests a concrete development plan.

The constraint system Sonic uses is the same as in the work by Bootle et al. [Boo+16], and its description can be found both there and in the Sonic paper itself. The constraints are more similar to R1CS than to QAP/SAP. While ZoKrates currently compiles its code to R1CS to then passes it to libsnark, there exist several possible implementation directions that modify this call stack; we suggest the following two:

1. Extending ZoKrates to convert arithmetic circuits to expressions in the Sonic constraint system. The conversion procedure is described in the work of Bootle et al, appendix A, and probably is the most straightforward solution to the problem.
2. Converting QAP/SAP or R1CS (which are the output of either ZoKrates or libsnark) to the constraint system of Sonic. This solution is less straightforward, and there are a few reasons for that. First, to our knowledge, there are no documented solutions explaining how to perform this conversion; R1CS to Sonic constraints converter was implemented for the original Sonic paper ⁷, but no verbal description of it is available. Second, the performance might be a problem: converting low-level representations might incur an overhead compared to the direct compilation from circuit description, and the precise performance of suggested original Sonic paper converter is unknown, although authors specify that “it adds some constant overhead during proving and verifying steps, but eases implementation and comparison with existing constructions”.

⁷<https://github.com/ebfull/sonic/blob/master/examples/paper.rs>

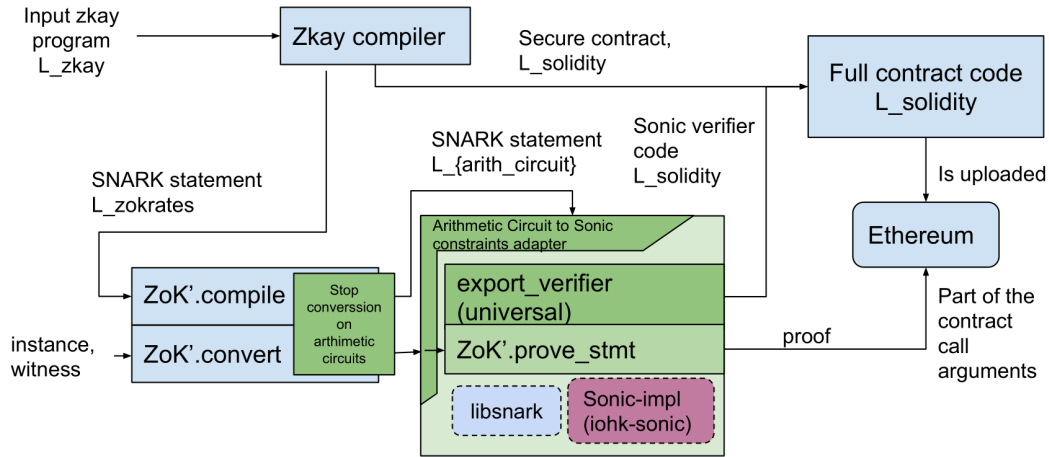


Figure 5.2: ZK toolkit, suggested solution 1 “Patching ZoKrates with arithmetic circuits to Sonic constraints converter”. In green things that are to be added/written, in purple – existing components that need to be imported and used, in blue – the old components. We remind that `export_verifier` can in theory be implemented outside of ZoKrates, and the fact it is presented here as being part of patched ZoKrates is for uniformity. This Sonic verification code in Solidity can be e.g. licensed under MIT and put into separate repository, and called from patched ZoKrates.

Additionally to the main task of converting zkay code into Sonic constraints, one should provide Sonic implementation in Solidity, as ZoKrates does currently for GM17. This is an independent task, but doing it as part of ZoKrates is a more uniform solution. We should note an important property of Sonic’s SRS, namely, that it is universal, so Solidity verifier’s code does not need to depend on the concrete verification circuit, and the SRS (limited only by the circuit size) can be written once, even separately from all the other code. This does not imply that choosing a particular set of parameters and generating the concrete SRS is a trivial task, but just that it can be done in parallel with other tasks.

The current state of the open source Sonic implementation (the protocol itself) comprises a few libraries (`ebfull/sonic`⁸, `adjoint-io/sonic`⁹, `LayerXcom/lx-sonic`¹⁰) which are themselves claiming to be experimental and work-in-progress. Moreover, there exists an implementation of Sonic currently being developed by IOHK (`input-output-hk/sonic`¹¹), and it needs to be explicitly mentioned that additionally to the public repository there also exists a private development effort around creating a standalone compiler for this sonic prover library which we will refer to as `iohk-sonic-compiler`. Another related resource is `matter-labs/alpha_line`¹² library which provides an MPC to set up a genesis Sonic SRS. The implementation by `adjoint-io` and `alpha_line` repository are in Haskell, and all the other mentioned public implementations are in Rust. Further communication should clarify which exact implementation shall be used in the final version of the toolkit; we suggest targeting IOHK’s implementation because of the potential inclusion of the aforementioned compiler in the development process, but in general the developed solution can kept independent of to the concrete Sonic implementation, since the NIZK’s API does not significantly depend on their particular details.

One additional issue we need to draw our attention to is licensing. Since the suggested development plan implies modifying, or building on ZoKrates, its LGPL licence might not be a suitable alternative, since its copyleft structure restricts us to using either LGPL or the full GPL only. Two other variants, in case the decision

⁸<https://github.com/ebfull/sonic>

⁹<https://github.com/adjoint-io/sonic>

¹⁰<https://github.com/LayerXcom/lx-sonic>

¹¹<https://github.com/input-output-hk/sonic>

¹²https://github.com/matter-labs/alpha_line

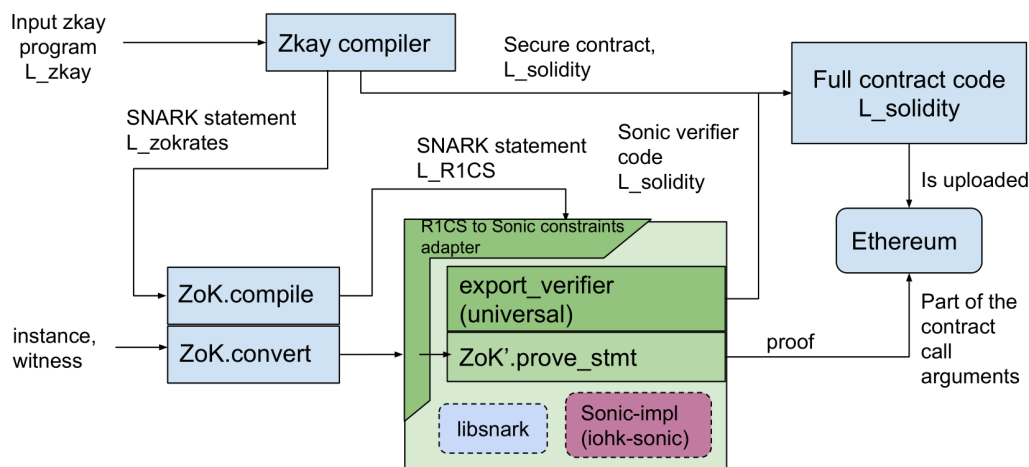


Figure 5.3: ZK toolkit, suggested solution 2 “Patching ZoKrates with R1CS to Sonic constraints converter”. We notice that the fact it doesn’t change the compilation and conversion part of ZoKrates doesn’t necessarily imply the solution is easier to implement.

is taken to avoid GPL, are available:

1. Implementing a standalone ZoKrates-like solution, that would compile high-level language description (Solidity code that zkay uses) into arithmetic circuits, and further into constraints. ZoKrates is a pretty massive project, with 1740 commits at the time of writing, and re-implementing it in a way that offers the same or better functionality might be a complicated goal and needs further estimation.
2. Converting zkay contract language directly to the input/intermediate representation languages iohk-sonic-compiler offers instead of converting them to ZoKrates input language. The complexity of this translation depends on multiple factors, most importantly on the specification language of the compiler. In the end, we also get all the optimisation the Sonic compiler provides, since they are happening on the lower, post-circuit level. This option suggests deeper integration with the IOHK toolstack, and requires further discussions, but from a technical perspective it seems to be more realistic.

Hence, for now omitting the licencing issue just described, the suggested first steps are to extend ZoKrates with the described additional compiler/translator, and to combine it with one of the Sonic implementations available. This can be done in two different ways: either by compiling intermediate arithmetic circuits produced by ZoKrates to Sonic constraints, as presented on the Figure 5.2, or implementing a R1CS to Sonic constraints converter, Figure 5.3. The third solution, which implies switching from Zokrates toolstack to the IOHK toolstack, is presented in the Figure 5.4. LGPL issue and possible options should be negotiated, as well as the state of their development process with respect to the private IOHK tools. Even in case some private components are not to be open-sourced immediately, at least their API and specification could be potentially disclosed to help start the (later phases of the) development process.

5.6 Future steps and extensions

A potential future steps would be to extend and redesign the zkay language itself by improving the expressiveness of private annotations, which could require more work on static program analysis and modification of the zkay code itself.

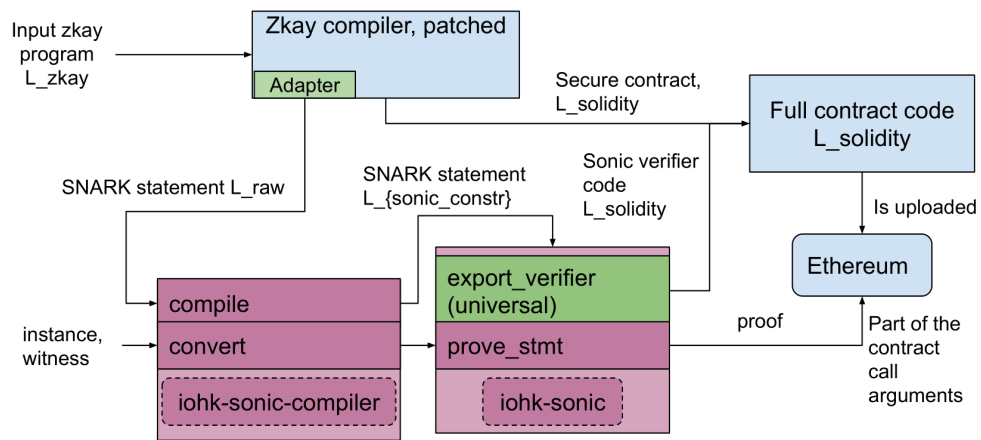


Figure 5.4: ZK toolkit, suggested solution 3 “Replacing ZoKrates with IOHK toolkit”. The Solidity verifier should still be implemented (as in any other solution), and zkay should export circuit description in iohk-sonic-compiler input format. All other components are used as external libraries.

Chapter 6

Toolkit for Ledger-Oriented Secure Two/Multi-Party Computation

The toolkit presented in this document offers secure multiparty computation protocols using a distributed ledger. The design of this toolkit consists of two components.

The first component is designed by UNISA and is a module that offers an interface to safely read and write on a blockchain. This module adds value to current libraries used to read/write on blockchains. First of all, it abstracts the use of a blockchain allowing to securely run on a generic blockchain protocol secure multi-party computations that were originally designed to run on point-to-point channels. With additional measures, such as proper message authentication and tagging, the protocol execution can be made publicly verifiable. This could enable attribution of protocol abortion, for example. Furthermore, the module could prevent security issues as a result of blockchain forking.

The module will support at least one popular permissionless blockchain and will offer features to maintain the security of the computation in presence of forks without penalizing excessively the performance.

The second module is designed by TUE and consists of an extension of MPyC [Ber18], a library for honest-but-curious secure multi-party computation in order to add verifiability of correct computation through zk-SNARKs.

6.1 Module 1: Abstract and Safe Communication with a Blockchain

This module focuses on two features:

1. network communication via a blockchain;
2. resilience of a protocol to fork attacks.

The first feature will be realized by developing a generic API that will allow reading and writing on a generic blockchain, abstracting the details of specific blockchains. The API will allow to open a blockchain, read and write data on a blockchain.

The toolkit exploits research results achieved in WP2/WP3 and presented in [Bot+19]. In that paper, the authors (two of which are affiliated with UNISA) prove that, in the presence of blockchain forks, rushing players (i.e., players that do not wait for confirmation before posting new transactions) might risk serious loss of privacy.

Security in the presence of forks. A straightforward way to obtain security in the presence of forks consists of always discarding unconfirmed blocks when reading from a blockchain. However, in some cases, it is safe and much faster to react immediately to unconfirmed transactions. The module will offer a default behaviour that secures the communication with a blockchain through delays waiting for block confirmation. Nevertheless,

the module will also offer a mechanism to override the above default behaviour allowing players to rush when this is considered safe.

API. The module will offer an API including the following functionalities:

- An **open blockchain** functionality that takes the blockchain identifier and a confirmation parameter and returns a handle to the blockchain. Opening a blockchain requires to specify the blockchain to use and a confirmation parameter that is a default value. The confirmation parameter is an optional value to pass to the functionality. It represents the number of blocks required to wait before assuming that the transaction is confirmed. If this value is not set then the library uses a default value.
- A **read** functionality that takes as input an identifier to recognize the transaction to read and a confirmation number. The module searches the transaction on the blockchain, and checks that the block including the transaction is followed by sufficiently many blocks.
- A **write** functionality that takes as input a message and stores it on the blockchain. This function is used to send a message to the blockchain. The library will not take into account the content of the message, it is able to write this message in a transaction and send it to the blockchain.
- A **setup** functionality used to modify the confirmation parameter. This functionality can be called at any moment to reset the confirmation parameter to the value that takes as input.

6.2 Module 2: Verifiable Honest-but-Curious Secure Multi-Party Computation

This module implements verifiable multiparty computation, as explained in Chapter 9 of PRIViLEDGE Deliverable 3.2 (“Design of Extended Core Protocols”). The module extends the MPyC library [Ber18] for secure multiparty computation in the honest-majority setting.

MPyC supports secure m -party computation tolerating a dishonest minority of up to t passively corrupt parties, where $m \geq 1$ and $0 \leq t \leq (m - 1)/2$. The underlying protocols are based on threshold secret sharing over finite fields (using Shamir’s threshold scheme as well as pseudorandom secret sharing).

Module 2 extends MPyC to provide verifiability. Verifiable MPC allows third parties (even parties who did not participate in the protocol) to verify the correctness of the result of the computation.

The functionality and interfaces of Module 2 are explained using the example of a private auction: Input parties post bids to a smart contract that outsources the secure computation of the auction result to a separate network of MPC parties. The MPC parties, who are external to the blockchain, download the private inputs from the blockchain (calling the specific smart contract), compute the auction result and post the result back to the smart contract. With verifiable MPC, this computation result is accompanied by a cryptographic proof of correctness, such that blockchain users can verify the validity of the outsourced computation. Module 2 extends the MPyC library to retrieve a cryptographic commitment or encrypted input from a smart contract, to convert the encryption to secret shares and to post the result with a proof of correctness back to the blockchain. A party that is external to the MPC computation can then independently verify the correctness of the computation. Figure 6.1 presents these functionalities in a schematic way.

The connection between pairs of MPC parties is implemented using client-server TLS connections. Optionally, Module 2 could use the functionalities of Module 1 to route MPC messages via the blockchain.

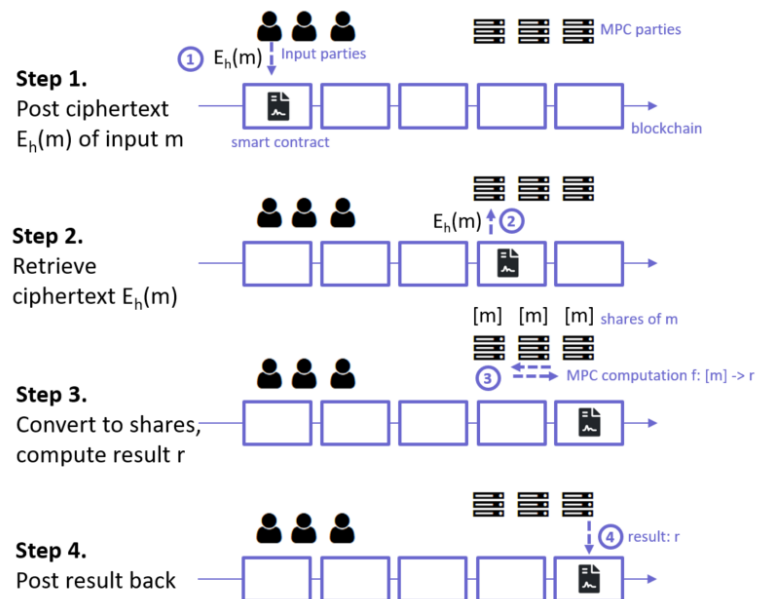


Figure 6.1: Schema illustrating input, retrieval, conversion and computation, and download functions of toolkit.

Chapter 7

Toolkit for Privacy-Preserving Data Storage on Ledgers

Before presenting PRIViLEDGE’s objectives to privacy-preserving data storage on ledgers, we’ll investigate the related state-of-the-art, treating the topic in two parts: techniques for privacy and security of the data and the transactions in the ledgers themselves (Section 7.1) and techniques for the safe storage of private data that is stored by ledger applications (Section 7.2).

7.1 Cryptographic Techniques for Security and Privacy in Ledgers

Most systems today aiming to provide privacy for the identities and the contents of transactions, as well as safe computation on private data, use some of the following techniques:

Mixing. Mixing provides preservation of privacy for identities by obfuscating transaction relationships on a ledger. There are two main ways to mix:

- *Mixing using third parties.* There are multiple mixing websites available that provide their services for some fee, however they don’t provide protection against coin theft. A well known implementation that achieves unlinkability of sender and receiver and avoids coin theft is TumbleBit [Hei+]. Although it requires a central server, the server has extremely limited power and no control over the funds, and it utilizes secure two party computation and zero-knowledge proofs.
- *Mixing with no need of a third party.* One method used is CoinJoin, which is compatible with blockchains similar to the Bitcoin platform and which ensures that even malicious nodes cannot find transaction details and link them with other transactions. A protocol that extends CoinJoin [conb] and increases privacy is CoinShuffle [RMK14]; it has been implemented in Shufflepuff [Dan15] and CashShuffle [Cas].

Ring Signatures. These are signature schemes that provide anonymity for the signer. In such schemes there is a group of possible signers and each member can provide a valid signature without revealing their identity, thus achieving anonymity. Some well-known privacy preservation protocols for ledgers that implement ring signatures are CryptoNote [Sab13], Monero [Mon], and Ethereum. Because CryptoNote is vulnerable to some attacks an improvement of it was proposed called Ring Confidential Transaction that is used by Monero. Monero also uses stealth addresses (see below) in order to hide the IP addresses of those interacting with the Monero network.

Stealth Addresses. Stealth addresses are used on a cryptocurrency network for hiding the recipient's address in a transaction. They are single-use addresses that can be generated using the elliptic-curve Diffie-Hellman protocol. Stealth addresses prevent the association of a receiver's public address, that is the actual destination address, with the recipient address of a public transaction, thus obscuring the actual receiver's identity and make it difficult to find the real amount of cryptocurrency that a user holds.

Non-Interactive Zero-Knowledge Proofs (NIZKs). A Non Interactive Zero-knowledge Proof is a cryptographic method in which a user, given some private inputs, can calculate an outcome, which is verifiable by a third party without the third party being aware about the private inputs. A NIZK can be used to provide anonymity for the sender in a transaction, such as in the Zerocoin [Mie+13] protocol, and it can also be used to hide the amounts sent in a transaction and provide verifiable outcomes of a computation on private data. A protocol that guarantees privacy of identity as well as the transaction amount is Zerocash, which implements Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) NIZKs and is used in the Zcash [Sas+14] cryptocurrency. Extending to the computation on private data, the ZoKrates toolkit [ET18a] provides verifiable proofs about the results of computations on secret inputs in the Ethereum platform. Zero Knowledge Scalable Transparent Argument of Knowledge (ZK-STARK) do away with the need for a trusted setup. An implementation of ZK-START is StarkDex [S0], used by decentralized exchanges, i.e., cryptocurrency exchanges that allow peer-to-peer trading of cryptocurrencies for settlements. Another variation of NIZKs that provides short proofs and requires no trusted setup are Bulletproofs [Bün+18], which are used in Bitcoin, Monero and recently by the ING Bank [Ban].

Secure Multi-Party Computation (MPC). An MPC is a cryptographic multi-party protocol that allows joint computations on private data, so that only the result is revealed without breaking the privacy of the input data. One of their first applications on blockchains was on the Bitcoin system for secure multi-party lotteries without the need of a trusted authority. More recently, Enigma [SSP18] utilizes MPC protocols in order to provide secure private computation on secret data. See also Chapter 6 for more on MPC and PRIViLEDGE.

Homomorphic Commitment Schemes. A commitment scheme allows committing to some value, without revealing the actual value. Homomorphic commitment schemes allow to multiply two commitments and get a new commitment that contains the product of the two messages. So, homomorphic commitment schemes allow to perform certain types of computations on ciphertexts while preserving the secrecy of the plaintext; an example is the Pedersen commitment scheme [Ped92]. Some examples where it is applied is in the construction of Bulletproofs [Bün+18], which are short NIZKs that don't require a trusted setup, and in the Internet of Things (IoT), where devices communicate with the ledger and there is a need of private computation on confidential data; such a confidential blockchain-enabled IoT system that supports fully homomorphic computation is Beekeeper [Zho+18].

Trusted Enclave Environments (TEEs). TEEs provide an isolated environment inside a host operating system, where the code of the smart contracts can be safely executed, without any software or party out of the enclave to be able to tamper or affect the computations or learn anything about the state of the execution. The most well-known TEE technology is Intel Software Guard eXtensions (SGX). Some examples of systems that utilize a TEE and are combined with ledgers are:

- *Town Crier* [Zha+16]. An authenticated data feed for smart contracts, Town Crier is a service that provides information about the outside world, such as data provided from a sensor placed somewhere in the outside world. By using a TEE the system ensures that the data provided to smart contracts for computations are safe. Town Crier can work as an oracle, providing answers to questions (requests) that cannot be tampered.
- *Ekiden*. In Ekiden [Che+19a; Che] the state of a contract and the data that will be used for safe execution are stored encrypted in the blockchain. The smart contracts themselves are privately executed inside a

TEE. During execution, the data needed as inputs are provided encrypted from the external storage and when the computation is complete, the results exit the TEE in encrypted form. The blockchain used in the system stores the keys and the encrypted information stored in a smart contract, more specifically the smart contract state.

- *Hyperledger Sawtooth* [Fouc; Ols+]. This project uses Intel’s SGX technology in its consensus mechanism, called Proof of Elapsed Time (PoET), to ensure that executions are performed in a safe environment. In the context of Hyperledger Sawtooth an example application to achieve privacy is the Private UTXO [Arc; Saw]. Two projects that have been proposed in order to achieve privacy of the transaction and consensus of the validators without revealing any transaction information are Private Data Objects [Laba], and Private Transaction Families [Labb].
- *PrivacyGuard*. PrivacyGuard is a prototype for the Ethereum platform with the following characteristics [Xia+19]:
 - The data are stored somewhere in a database or maybe in the cloud.
 - Smart contracts exist in the blockchain and hold information about access policies for the data.
 - All computation on the data are executed off-chain in trusted enclaves according to the access policies.
- *ShadowEth*. ShadowEth [Yua+18] aims at privately executing smart contracts without someone being able to find details about their code, inputs and outputs. The contract that should be private is stored inside an enclave and when a node of the system needs to execute it, it communicates with the enclave, inside a secure channel, takes the encrypted data and consequently it executes them inside its enclave. Then the metadata of the private execution that certify the validity of the computations are stored in a smart contract that lives in blockchain.
- *DeepLinQ*. The project is still under development and it is proposed in the paper [Cha+18]. The system is multi-layered, consists of two blockchains and has initially been applied in medical databases for medical data. One of the two blockchains, called basic, is used to store Merkle hashes of the transactions executed in the upper layer. The basic blockchain can be the Ethereum ledger. The upper layer stores smart contracts that define access control for private data stored in medical databases.

Note that TEEs are not invulnerable to attacks; e.g., see [Sch+17; Jan+17; Göt+17; Van+18; Dal+18].

7.2 Safe Storage of Private Data Used on Ledgers

Blockchain platforms should not be used to store big files, as the nodes of the network would have to store more data necessary to achieve consensus, so some other decentralized applications have been created for secure and private storage of data, not in a centralized server, that can be used in parallel with a ledger. Some of such systems are listed below:

- *InterPlanetary File System (IPFS)* [Labc]. When a file is stored in IPFS, a hash value is calculated for the file. Then other nodes of the network may choose to store the content of the file. Later, when users want to retrieve the content of file they can trace it using the hash value that previously was calculated for the specific file. To provide privacy of the data the file can be encrypted before it is stored so that it can be decrypted by specific users. A drawback is that the nodes of the network choose which files to store. So, if users want to permanently store data on the network, they have to use a pinning service, which is a payable service that operates some IPFS nodes and guarantees to keep the data on the network. A different approach is taken by the Filecoin project [Fila; Lab17]. The idea behind Filecoin is to incentivize nodes to provide their free space and store specific data, thus solving the key problem of permanently storing data

in IPFS. Filecoin will implement Proof of Replication for verifying the storage of a file and proof of space time so that a user can check if a storage provider is storing the data at the time of the challenge. Filecoin will also use a coin. The IPFS implementation is at [IPF] and the Filecoin implementation at [Filb].

- *Social Linked Data or Solid* [Sol]. Solid is a web decentralization project whose primary goal is to give users the possibility to own their own data and define where they will be stored and who will have access to them. It aims to build a platform for applications that will use linked data. Users can have a place, called a Solid Pod, to store their data. Someone can set up a server and use it as their Solid Pod or can choose an online Solid Pod provider to store them. When a service needs information about users, it will be redirected to their Pod provider for authentication and any data the service will need, will be fetched from the Pod after the authentication step is passed. Solid currently uses WebID-TLS [ISH] and WebID-OIDC, that is based on the standard OpenID Connect (OIDC) [Foub] and is used by decentralized systems such as Solid that use WebIDs, for the authentication mechanism. A WebID is an identity spec that allows a URI to denote a user and point the place from where user's data can be retrieved. Solid could integrate with IPFS and run in a layer above it.
- *Swarm* [FUN18]. This is a distributed storage platform and content-addressable P2P storage service and is designed to integrate with the Ethereum ledger. When users want to upload content, they first upload the data to their local Swarm node; then, the local Swarm node syncs the chunks of data with peers in the network. For downloading content, the local Swarm node queries the peers in the network for the relevant chunks of data and then reassembles the content locally. The users can opt for encryption of data. Swarm uses Counter mode encryption (CTR) for encryption and decryption. When data are uploaded to a local Swarm node they are split into 4 KB chunks. These chunks will all be encoded with a separate randomly generated encryption key. The encryption takes place on the local Swarm node and then the encrypted data are shared to the peers. In this way other nodes in the network are not able to access the original data, or distinguish if the chunks are encrypted or not. Data can be retrieved using an identifier that is produced from their content, so that if the data are updated the identifier will change. To provide a persistent identifier for data, a feature called Mutable Resource Updates has been designed, which will connect a key to a resource at creation time. Like IPFS, Swarm has not yet implemented an incentive method for nodes to permanently store specific data.

The Ethereum Foundation operates a Swarm testnet that can be used to test out functionality [Ethb]. Decentralized applications can interact with Swarm by using a local HTTP proxy API provided by Swarm. Everyone can join the network by running the Swarm client node. Also public gateways are hosted by the Ethereum Foundation providing free access to users that want to try Swarm without running their own node.

- *MaidSafe or SAFE network* [Maia; Net; For]. The SAFE network is an open source, decentralized network for storing data in spare computing resources of the network's users. In particular, users can provide their computer resources and get paid in network's token called Safecoin. When data is uploaded to the network they are first split into pieces and then each piece is encrypted using self-encryption (a version of convergent encryption with an additional obfuscation step) [Maib]. The encrypted chunks are then stored at random locations known only to the network. As the data is encrypted before being uploaded to the network, only the user who is the owner of the data can access them, unless they choose to share or publish it in public. In order to confirm a Safecoin transaction when a user stores a piece of data, SAFE implements the PARSEC consensus protocol.
- *Sia* [Siab; VC14; Siao]. Sia provides decentralized cloud storage and aims to connect users who need file storage with hosts worldwide that offer their hard drive capacity. Data are split using Reed-Solomon erasure coding to ensure that they can be recovered even if a number of hosts is offline. The file segments are encrypted with Twofish encryption before they leave the renter's computer. Users that want to store data first form a file contract with a host. These contracts are a type of smart contract that implement

agreements that are stored on the Sia blockchain. Renters and hosts use Siacoin, a cryptocurrency built on the Sia blockchain. Hosts put coins as collateral to a contract so they are disincentivized to go offline, thus helping to achieve permanent data storage. Micropayments between renters and hosts are done using payment channels, that are similar to Bitcoin's Lightning Network. When a contract is concluded, a percentage of the deposit is paid to holders using a secondary token, Siafunds (90% of Siafunds are held by the developers of Sia). The host must prove that they store the renter's data by providing a storage proof that is created at the end of a contract. The storage proof must appear on the blockchain within a certain timeframe for the host to get paid otherwise the host is penalized. A proof's size is small, which is important since proofs are permanently stored in the blockchain.

A difference with Swarm and IPFS is that Sia doesn't focus on creating an infrastructure to replace HTTP, as in Swarm and IPFS where data are retrieved with an identifier relevant to their content. Moreover, user data can be deleted after being shared with a host, and as users pay the hosts the latter are incentivized to permanently store data.

- *Dat* [Foua; OMM17; Pro; Kea]. *Dat* is a project that aims to store, share, and track large files securely without relying on a central service like Dropbox. It uses a peer-to-peer (P2P) network like BitTorrent, but in contrast with BitTorrent that is used to serve static files, *Dat* is able to update and sync files over the P2P network. For now, only one user can write into an archive and that is the original creator of the *Dat* who is given a keypair. This keypair can be used for verification so that only the creator is allowed to add or update files in the *Dat*. If the keypair is lost the data can no longer be updated.
- *Storj* [Inc; Sto18]. *Storj* is an open source project for storing data in a decentralized network. It has some similarities with BitTorrent and runs on a peer-to-peer network. Some of its key components are the use of:
 - Encryption; files are encrypted in the user's computer before they are stored to the network.
 - File sharding; data are broken into smaller pieces before they are stored.
 - A blockchain-based hash table, called Kademlia, to store files on the network. Recently *Storj* integrated with the Ethereum blockchain to host its application and hash table.

A key difference between Torrent implementations and *Storj* is that the former make public the location of the shards, while in the latter only the owner of the data knows where the shards of their file are stored. To achieve this, *Storj* uses a distributed hash table so users can locate all the shards of their original file, using a private key. Besides the open source *Storj* software that anyone can use to run an instance of it, *Storj Labs*, a for-profit company, has a network of thousand of users that run *Storj* software and it charges for the use of its network [Labd].

7.3 Privacy-Preserving Ledger Storage Toolkit

This section aims to provide a high level description of the toolkit for privacy-preserving data storage on ledgers. If possible the toolkit will be exploited by PRIViLEDGE use cases, such as, in particular, UC3: University Diploma Record Ledger.

For the development of the toolkit, it should first be clear what data will be stored on the ledger. Generally, a good practice is to avoid storing bulk data on ledgers. In this way, the need for large free space to maintain a copy of the ledger is decreased and more nodes of the network are able to store the whole ledger. This is important as these nodes, which are called full nodes, are responsible for the validation process in the transactions, so their number should be large enough to avoid centralization. To gain some insight, the size of the Bitcoin blockchain is about 252 GB [com] and of the Ethereum blockchain is about 4 TB [Eth].

Moreover, off-chain storage can offer enhanced privacy and security. Even if the data are stored encrypted or encoded on the ledger, adversaries may find ways to violate privacy and find information about the plaintext. As

an example, if the ledger stores the hash value of a diploma, someone could guess the plaintext, then generate its hash value and finally compare it to the one stored to the blockchain. In case they are identical, the guess is confirmed.

An additional reason for not saving a large size of data on ledger is the design limitations of blockchains. For example, the block size for Bitcoin is specified in bytes and it is 1 MB [cona]. That number is enough to store information regarding transactions, but not enough for bigger files.

Currently most of the applications that exploit the benefits of DLTs store data:

- mainly to an external storage system (centralized or decentralized)
- rarely on the ledgers themselves, if the amount of data is small (e.g., an encrypted hash value)

Moreover the data usually stored on ledgers are transactions, commitments and metadata, not the actual data per se. The strategy most often used is to store small volumes of data on ledgers that link to actual data residing elsewhere, i.e., to an external database that is either centralized or decentralized.

Note that even if the actual data are saved in a database, it is often advantageous to integrate DLTs in an application. The reason is that DLTs have some crucial advantages like the immutability and verifiability of the data stored and distributed consensus which helps keeping an uncorrupted sequence of blocks. These benefits assist applications that interact with ledgers to achieve:

- Enhanced user-control over their data. Blockchains can set access control rules defining who is eligible to write on their ledger. In permissioned blockchains, only users who have the required right can write data in them. Also some blockchains support the execution of smart contracts on their platform. Smart contracts are code that can be designed in such a way to allow execution of an action only if specific requirements are met. So for example if the ledger is used to store a value that points to an external database, which holds personal information about a user, it is difficult to modify the value without being noticed. One way to be aware of the modification, if the value is encoded on the transaction, is to inspect the transactions that are recorded on the ledger and observe what data they carry. Alternatively, if the value is stored inside a smart contract and changes by interacting with one of its functions, we can emit an event when the function executes, so an application can set up a listener function for that event. Also the user could periodically read the ledger and try to access the database that the value points. So the user will always be aware where their data are stored. When centralized databases are used without integration with blockchains, companies may move users' data to another database unnoticed, but in blockchains every action is recorded immutably.
- Accountability and verifiability of the data kept in the ledgers, as every step is recorded. By inspecting a blockchain transaction someone can find out who initiated every action and thus who is accountable for it.
- Version control. Anyone that is able to read the ledger can find out what is the current value of a data that is stored on it and which were its previous versions.

While centralized databases have benefits, such as low latency, high speed, and quick throughput, at the same time they have some major disadvantages mostly because the data are stored in one specific place. So the data can be easily hacked, breached, and also effortlessly gathered and moved to another place. The last point means that effectively users have no control over the location of their data.

A solution to overcome the drawbacks of centralized databases and to achieve better privacy for the stored data is to keep the data in a decentralized manner. As we have seen, most decentralized approaches use end-to-end encryption and sharding to preserve the privacy of users' data. More specifically, usually before data are distributed, they are split into chunks, then they are encrypted, and then they are sent to different nodes of a storage network. For data retrieval one must be in possession of the encryption's private key in order to decrypt the files. In most systems distributed tables with references are used to track the pieces of the data for retrieval.

In addition some systems give to their users the freedom to choose which nodes they want to store their files, so they are more in control of their own data.

Today, one of the most well known protocols for data distribution is IPFS and some platforms that could be used for distributed storage and for content sharing are Swarm, Storj, Maidsafe or Safenetwork, Sia, which we have briefly described. For the development of the toolkit, the previously mentioned keypoints are taken into account. Some of the key features that the toolkit is desirable to provide are:

- Communication with the ledger and with the service. The toolkit must act as a bridge between the ledger and the service.
- Read and write operations on the ledger.
- Return proofs. The toolkit should give:
 - A zero-knowledge proof. This proof will persuade the verifier that the claim stored in the ledger is correlated with the proof in the (external) database, without leaking any information or allowing for chosen-plaintext attacks. For example, if verifiers know that the ledger contains information that points to databases that store the actual data, then when a zero knowledge proof is returned to them from the toolkit, that correlates the information on the ledger with the data on an external storage, they can be certain that the data in the external database are legitimate.
 - A receipt, that acts as a proof, when data are successfully stored in the ledger.
- Enhanced user control. Users should be notified and give their consent before the execution of any operation that is associated with their data. Also users can opt to store their data to an external database of their choice. The toolkit may assist them accomplish this storage operation or the user can complete the process on their own and just provide the necessary information to the toolkit.

Following the above, the designed toolkit overview architecture is illustrated in Figure 7.1.

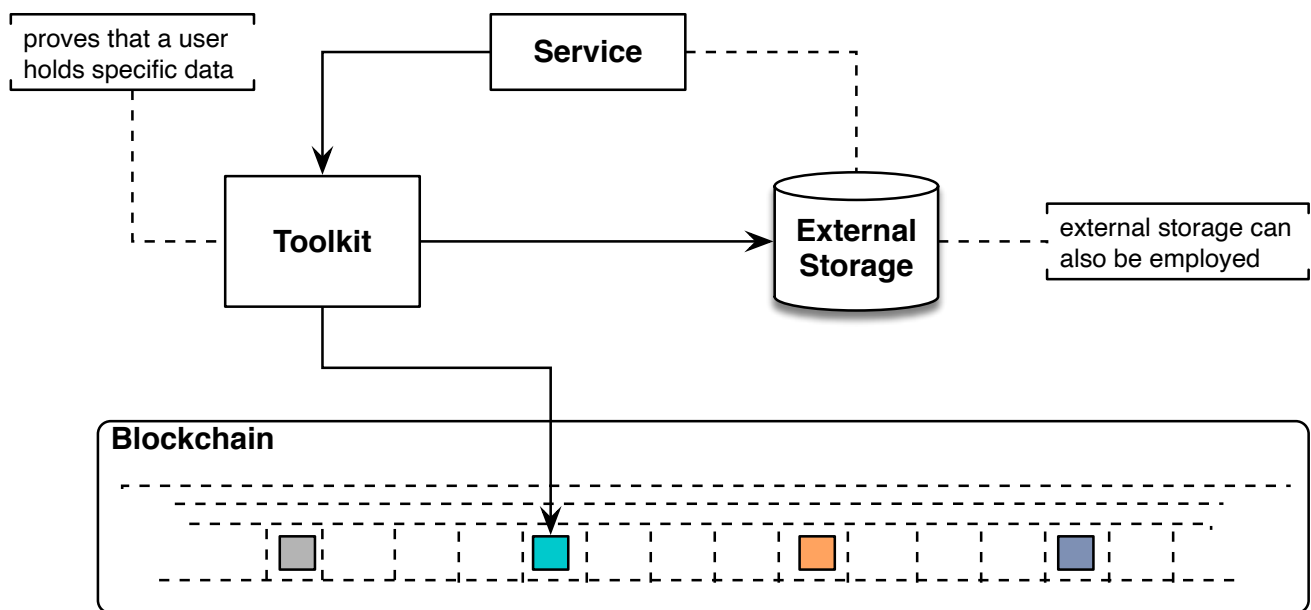


Figure 7.1: Storage toolkit overview architecture.

The toolkit will offer a generic API that will allow to read and write data on a ledger and also read data from an external database. For now it is not yet decided the specific blockchain platform that will be used for the communication, but preferably it would be one that supports smart contracts. A more detailed description of the API functionalities are:

- A write operation that takes three inputs. One of them is an identifier, the second is a message, such as an encoded URL that will be used to retrieve data from a database, and the third will be a proof that correlates the data stored in the database that the second input points to, with the identifier given as a first input. After the successful execution of the operation, the ledger returns to the toolkit a receipt that data are successfully stored in the ledger.
- A read operation that takes as input an identifier. Based on that identifier the toolkit will search the ledger and retrieve data. The data will later be used in the read functionality, executed on an external database.
- A read operation to an external database, that takes as input a message, in the form of a URL, and makes a request to an external storage in order to retrieve data.

The toolkit will be generic so that it will be usable in different contexts; in PRIViLEDGE, the toolkit could be used to offer secure storage for data in UC3, as shown in Figure 7.2. The DIPLOMATA protocol would use the same toolkit to access and store data either on the ledger or off the ledger; recall that the protocol requires all protocol steps to be stored on the ledger. Other data, such as QUALIFICATIONS themselves, which are outside the main flow of the protocol, would be stored off the ledger.

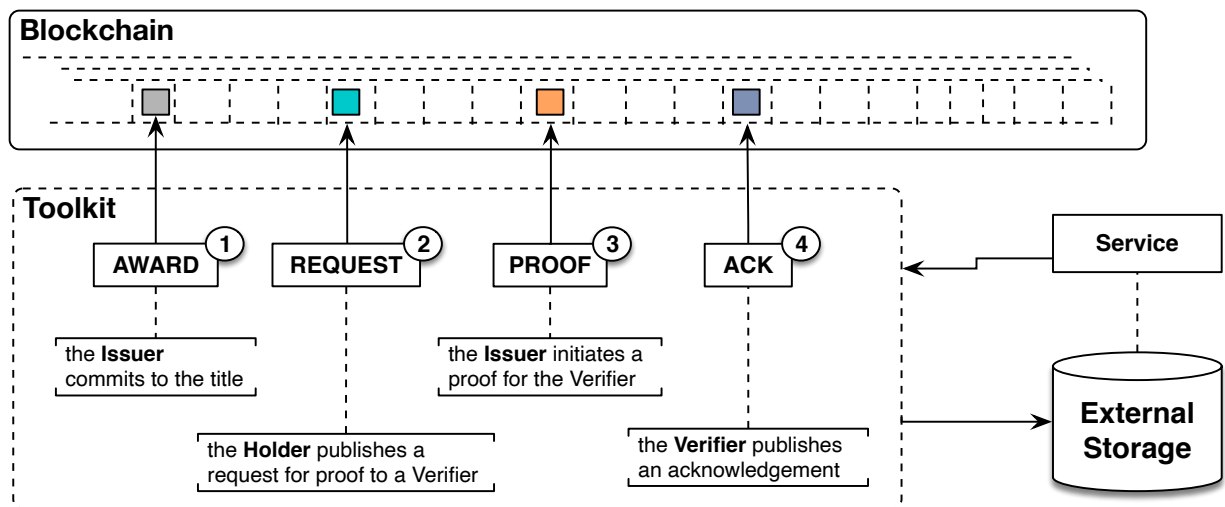


Figure 7.2: Storage toolkit in the University Diplomas Use Case.

Chapter 8

Conclusions

The use cases and the toolkits presented in this deliverable cover a wide area of applications and enabling technologies. PRIViLEDGE will now proceed to realising the use cases and the toolkits, where progress has already been made. This will require a dialogue between the architectures described in this document and the feedback from their implementation. As the technologies involved are rapidly evolving, it will also demand keep tracking of emerging trends and advancements in theory as well as in practice. Of particular interest will be the cooperation of research and engineering partners as, on the one hand, PRIViLEDGE is based on solid theoretical work, while, on the other hand, translating research to working systems requires sedulous technical effort. The next steps in the project, therefore, will combine both research and practice, in order to refine and in the end fulfill the plans outlined in this high-level deliverable.

Bibliography

- [Abd+17] Behzad Abdolmaleki et al. “A Subversion-Resistant SNARK”. In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10626. Lecture Notes in Computer Science. Springer, 2017, pp. 3–33. DOI: 10.1007/978-3-319-70700-6_1. URL: https://doi.org/10.1007/978-3-319-70700-6_1.
- [Arc] Hyperledger Project Archive. *Private UTXO*. URL: <https://github.com/hyperledger-archives/sawtooth-private-utxo> (visited on 06/12/2019).
- [Avi19] Gennaro Avitabile. “End-To-End Verifiable Internet Voting on Permissioned Blockchains”. University of Salerno, 2019.
- [Ban] ING Bank. *Reusable library for creating and verifying zero-knowledge range proofs and set membership proofs*. URL: <https://github.com/ing-bank/zkrp> (visited on 06/12/2019).
- [Ben07] Josh Benaloh. “Ballot Casting Assurance via Voter-Initiated Poll Station Auditing”. In: *2007 USENIX/AC-CURATE Electronic Voting Technology Workshop, EVT’07, Boston, MA, USA, August 6, 2007*. Ed. by Ray Martinez and David A. Wagner. USENIX Association, 2007. URL: <https://www.usenix.org/conference/evt-07/ballot-casting-assurance-voter-initiated-poll-station-auditing>.
- [Ber18] Berry Schoenmakers. *MPyC: Secure Multiparty Computation in Python*. 2018. URL: <https://github.com/lschoe/mpyc>.
- [Boo+16] Jonathan Bootle et al. “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 327–357. DOI: 10.1007/978-3-662-49896-5_12. URL: https://doi.org/10.1007/978-3-662-49896-5_12.
- [Bot+19] Vincenzo Botta et al. *Secure and Fair Computations on Forking Blockchains*. Cryptology ePrint Archive, Report 2019/891. <https://eprint.iacr.org/2019/891>. 2019.
- [Bow+18] Sean Bowe et al. “Zexe: Enabling Decentralized Private Computation”. In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 962. URL: <https://eprint.iacr.org/2018/962>.
- [Bün+18] Benedikt Bünz et al. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020. URL: <https://doi.org/10.1109/SP.2018.00020>.
- [Bün+19] Benedikt Bünz et al. “Zether: Towards Privacy in a Smart Contract World”. In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 191. URL: <https://eprint.iacr.org/2019/191>.
- [Cas] CashShuffle. *CashShuffle website*. URL: <https://cashshuffle.com/> (visited on 05/26/2020).

- [Cha+18] E. Y. Chang et al. “DeepLinQ: Distributed Multi-Layer Ledgers for Privacy-Preserving Data Sharing”. In: *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2018, pp. 173–178. DOI: 10.1109/AIVR.2018.00037. URL: <https://doi.ieeeecomputersociety.org/10.1109/AIVR.2018.00037>.
- [Che] Raymond Cheng. *Ekiden repository*. URL: <https://github.com/ekiden> (visited on 06/12/2019).
- [Che+19a] R. Cheng et al. “Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts”. In: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 2019, pp. 185–200.
- [Che+19b] Raymond Cheng et al. “Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts”. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 185–200. DOI: 10.1109/EuroSP.2019.00023. URL: <https://doi.org/10.1109/EuroSP.2019.00023>.
- [com] BLOCKCHAIN company. *Chart of bitcoin blockchain size*. URL: <https://www.blockchain.com/en/charts/blocks-size> (visited on 05/12/2019).
- [cona] Bitcoin wiki contributors. *Block size limit controversy*. URL: https://en.bitcoin.it/wiki/Block_size_limit_controversy (visited on 05/12/2019).
- [conb] Bitcoin wiki contributors. *CoinJoin wiki information*. URL: <https://en.bitcoin.it/wiki/CoinJoin> (visited on 05/26/2020).
- [Dal+18] Fergus Dall et al. “CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems 2018.2* (May 2018), pp. 171–191. DOI: 10.13154/tches.v2018.i2.171-191. URL: <https://tches.iacr.org/index.php/TCHES/article/view/879>.
- [Dan15] Daniel Krawisz. *Shufflepuff: A Mycelium implementation of CoinShuffle*. 2015. URL: <https://libraries.io/github/DanielKrawisz/Shufflepuff>.
- [ET18a] J. Eberhardt and S. Tai. “ZoKrates - Scalable Privacy-Preserving Off-Chain Computations”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCoM) and IEEE Smart Data (SmartData)*. 2018, pp. 1084–1091.
- [ET18b] Jacob Eberhardt and Stefan Tai. “ZoKrates - Scalable Privacy-Preserving Off-Chain Computations”. In: *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCoM) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCoM/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018*. IEEE, 2018, pp. 1084–1091. DOI: 10.1109/Cybermatics__2018.2018.00199. URL: https://doi.org/10.1109/Cybermatics%5C_2018.2018.00199.
- [Etha] Ethereum. *Chart of ethereum blockchain size*. URL: <https://etherscan.io/chart2/chaindatasizefast> (visited on 05/12/2019).
- [Ethb] Ethersphere. *Swarm: Censorship resistant storage and communication infrastructure for a truly sovereign digital society* <https://swarm.ethereum.org/>. URL: <https://github.com/ethersphere/swarm> (visited on 06/12/2019).
- [Fila] Filecoin. *Filecoin website*. URL: <https://filecoin.io/> (visited on 06/12/2019).
- [Filb] Filecoin. *Github repository of Filecoin project*. URL: <https://github.com/filecoin-project> (visited on 06/12/2019).

- [For] SAFE Network Forum. *SAFE Network Fundamentals: Context*. URL: <https://safenetforum.org/t/safe-network-fundamentals-context/25352> (visited on 06/12/2019).
- [Foua] Dat Foundation. *Dat website*. URL: <https://dat.foundation/> (visited on 05/26/2020).
- [Foub] OpenID Foundation. *Description of OIDC by OpenID Foundation*. URL: <https://openid.net/connect/> (visited on 05/26/2020).
- [Fouc] The Linux Foundation. *Hyperledger Sawtooth project*. URL: <https://www.hyperledger.org/use/sawtooth> (visited on 06/12/2019).
- [FUN18] SWARM FUND. *Swarm whitepaper*. Version 0.91. 2018. URL: <https://docs.swarm.fund/swarm-whitepaper-eng.pdf> (visited on 05/26/2020).
- [GM17] Jens Groth and Mary Maller. “Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs”. In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. Lecture Notes in Computer Science. Springer, 2017, pp. 581–612. DOI: 10.1007/978-3-319-63715-0_20. URL: https://doi.org/10.1007/978-3-319-63715-0%5C_20.
- [Göt+17] Johannes Götzfried et al. “Cache Attacks on Intel SGX”. In: *Proceedings of the 10th European Workshop on Systems Security*. EuroSec’17. Belgrade, Serbia: Association for Computing Machinery, 2017. ISBN: 9781450349352. DOI: 10.1145/3065913.3065915. URL: <https://doi.org/10.1145/3065913.3065915>.
- [Gro+18] Jens Groth et al. “Updatable and Universal Common Reference Strings with Applications to zk-SNARKs”. In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*. 2018, pp. 698–728. DOI: 10.1007/978-3-319-96878-0_24. URL: https://doi.org/10.1007/978-3-319-96878-0%5C_24.
- [Gro16] Jens Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11. URL: https://doi.org/10.1007/978-3-662-49896-5%5C_11.
- [Hei+] Ethan Heilman et al. *TumbleBit*. URL: <https://cs-people.bu.edu/heilman/tumblebit/> (visited on 05/26/2020).
- [HW14] Sven Heiberg and Jan Willemson. “Verifiable internet voting in Estonia”. In: *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*. Ed. by Robert Krimmer and Melanie Volkamer. IEEE, 2014, pp. 1–8. DOI: 10.1109/EVOTE.2014.7001135. URL: <https://doi.org/10.1109/EVOTE.2014.7001135>.
- [Inc] Storj Labs Inc. *Storj website*. URL: <https://storj.io/> (visited on 05/26/2020).
- [IPF] IPFS. *Peer-to-peer hypermedia protocol*. URL: <https://github.com/ipfs/ipfs> (visited on 06/12/2019).
- [ISH] Toby Inkster, Henry Story, and Bruno Harbulot. *W3C standard WebID-TLS specification*. URL: <https://dvcs.w3.org/hg/WebID/raw-file/tip/spec/tls-respec.html> (visited on 05/26/2020).
- [Jan+17] Yeongjin Jang et al. “SGX-Bomb: Locking Down the Processor via Rowhammer Attack”. In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. SysTEX’17. Shanghai, China: Association for Computing Machinery, 2017. ISBN: 9781450350976. DOI: 10.1145/3152701.3152709. URL: <https://doi.org/10.1145/3152701.3152709>.

- [JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. “Coercion-resistant electronic elections”. In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*. Ed. by Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine. ACM, 2005, pp. 61–70. DOI: 10.1145/1102199.1102213. URL: <https://doi.org/10.1145/1102199.1102213>.
- [Kal+18] Harry A. Kalodner et al. “Arbitrum: Scalable, private smart contracts”. In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, 2018, pp. 1353–1370. URL: <https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner>.
- [Kea] Duncan Keall. *How Dat Works*. URL: <https://datprotocol.github.io/how-dat-works/> (visited on 06/12/2019).
- [KKK20] Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. “Mining for Privacy: How to Bootstrap a Snarky Blockchain”. In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 401. URL: <https://eprint.iacr.org/2020/401>.
- [Kos+15] Ahmed E. Kosba et al. “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts”. In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 675. URL: <http://eprint.iacr.org/2015/675>.
- [Laba] Hyperledger Labs. *Private Data Objects*. URL: <https://github.com/hyperledger-labs/private-data-objects> (visited on 06/12/2019).
- [Labb] Hyperledger Labs. *Private Transaction Families*. URL: <https://github.com/hyperledger-labs/private-transaction-families> (visited on 06/12/2019).
- [Labc] Protocol Labs. *Ipfs website*. URL: <https://ipfs.io/> (visited on 06/12/2019).
- [Labd] Storj Labs. *Github repository of Storj project*. URL: <https://github.com/storj/storj> (visited on 06/12/2019).
- [Lab17] Protocol Labs. *Filecoin: A Decentralized Storage Network*. 2017. URL: <https://filecoin.io/filecoin.pdf> (visited on 05/29/2020).
- [Maia] MaidSafe. *Github repository of Maidsafe*. URL: <https://github.com/maidsafe> (visited on 06/12/2019).
- [Maib] MaidSafe. *self_encryption*. URL: <https://github.com/maidsafe/self-encryption> (visited on 06/18/2020).
- [Mal+19] Mary Maller et al. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. 2019, pp. 2111–2128. DOI: 10.1145/3319535.3339817. URL: <https://doi.org/10.1145/3319535.3339817>.
- [Mie+13] I. Miers et al. “Zerocoin: Anonymous Distributed E-Cash from Bitcoin”. In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 397–411.
- [Mon] Monero. *Monero research lab*. URL: <https://web.getmonero.org/resources/research-lab/> (visited on 05/26/2020).
- [Net] SAFE Network. *SAFE Network DevHub*. URL: <https://hub.safedev.org/> (visited on 06/12/2019).
- [Ols+] Kelly Olson et al. *Hyperledger Sawtooth whitepaper*. URL: https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf (visited on 06/12/2019).

- [OMM17] Maxwell Ogden, Karissa McKelvey, and Mathias Buus Madsen. *Dat—Distributed Dataset Synchronization And Versioning*. Apr. 2017. DOI: 10.31219/osf.io/nsv2c. URL: osf.io/nsv2c.
- [Ped92] Torben Pryds Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *Advances in Cryptology — CRYPTO ’91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140. ISBN: 978-3-540-46766-3.
- [Per] Oliver Pereira. *Verifiable Elections with Commitment Consistent Encryption A Primer*. <https://arxiv.org/pdf/1412.7358.pdf>. (01.04.2020).
- [Pro] Dat Project. *Github repository of Dat Project*. URL: <https://github.com/datproject> (visited on 06/12/2019).
- [RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. “CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin”. In: *Computer Security - ESORICS 2014*. Ed. by Mirosław Kutylowski and Jaideep Vaidya. Cham: Springer International Publishing, 2014, pp. 345–364.
- [S0] StarkWare and 0x. *StarkDex website*. URL: <https://www.starkdex.io/> (visited on 05/26/2020).
- [Sab13] Nicolas van Saberhagen. *CryptoNote v 2.0*. 2013.
- [Sas+14] E. B. Sasson et al. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *2014 IEEE Symposium on Security and Privacy*. 2014, pp. 459–474.
- [Saw] Hyperledger Sawtooth. *Hyperledger Sawtooth reference of Private UTXO in documentation*. URL: <https://sawtooth.hyperledger.org/docs/core/nightly/1-1/introduction.html#real-world-application-examples> (visited on 05/25/2020).
- [Sch+17] Michael Schwarz et al. “Malware Guard Extension: Using SGX to Conceal Cache Attacks”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Michalis Polychronakis and Michael Meier. Cham: Springer International Publishing, 2017, pp. 3–24. ISBN: 978-3-319-60876-1.
- [Siaa] Sia. *Gitlab repository of Sia*. URL: <https://gitlab.com/NebulousLabs/Sia/tree/master> (visited on 06/12/2019).
- [Siab] Sia. *Sia website*. URL: <https://sia.tech/> (visited on 05/26/2020).
- [Sol] Solid. *Solidproject website*. URL: <https://solidproject.org/> (visited on 06/12/2019).
- [SSP18] H. Shrobe, D. L. Shrier, and A. Pentland. “Chapter 15 Enigma: Decentralized Computation Platform with Guaranteed Privacy”. In: *New Solutions for Cybersecurity*. 2018, pp. 425–454.
- [Ste+19] Samuel Steffen et al. “zkay: Specifying and Enforcing Data Privacy in Smart Contracts”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro et al. ACM, 2019, pp. 1759–1776. DOI: 10.1145/3319535.3363222. URL: <https://doi.org/10.1145/3319535.3363222>.
- [Sto18] Inc Storj Labs. *Storj whitepaper*. 2018. URL: <https://storj.io/storj.pdf> (visited on 05/26/2020).
- [Van+18] Jo Van Bulck et al. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient out-of-Order Execution”. In: *Proceedings of the 27th USENIX Conference on Security Symposium, SEC’18*. Baltimore, MD, USA: USENIX Association, 2018, pp. 991–1008. ISBN: 9781931971461.
- [VC14] David Vorick and Luke Champine. *Sia whitepaper*. 2014. URL: <https://sia.tech/sia.pdf> (visited on 05/26/2020).

- [W3C19] W3C Working Group Note. *Verifiable Credentials Implementation Guidelines 1.0*. <https://www.w3.org/TR/vc-imp-guide/>. 2019.
- [W3C20] W3C Working Group Note. *Decentralized Identifiers (DIDs) v1.0*. <https://www.w3.org/TR/2020/WD-did-core-20200625/>. 2020.
- [Xia+19] Yang Xiao et al. *Enforcing Private Data Usage Control with Blockchain and Attested Off-chain Contract Execution*. 2019. arXiv: 1904.07275 [cs.CR].
- [Yua+18] Rui Yuan et al. “ShadowEth: Private Smart Contract on Public Blockchain”. In: *Journal of Computer Science and Technology* 33.3 (May 2018), pp. 542–556. DOI: 10.1007/s11390-018-1839-y. URL: <https://doi.org/10.1007/s11390-018-1839-y>.
- [Zha+16] Fan Zhang et al. “Town Crier: An Authenticated Data Feed for Smart Contracts”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 270–282. ISBN: 9781450341394. DOI: 10.1145/2976749.2978326. URL: <https://doi.org/10.1145/2976749.2978326>.
- [Zho+18] Lijing Zhou et al. “BeeKeeper 2.0: Confidential Blockchain-Enabled IoT System with Fully Homomorphic Computation”. In: *Sensors* 18 (Nov. 2018), p. 3785. DOI: 10.3390/s18113785.