



DS-06-2017: Cybersecurity PPP: Cryptography


PRIViLEDGE
Privacy-Enhancing Cryptography in Distributed Ledgers

D1.1 – Requirements and Interface Design

Due date of deliverable: 31 December 2018
Actual submission date: 31 December 2018

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

Lead contractor: Guardtime AS
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	

D1.1

Requirements and Interface Design

Editor

Panos Louridas (GRNET)

Contributors

Panos Louridas (GRNET), Dimitris Mitropoulos (GRNET), Georgios Tsoukalas (GRNET)

Sven Heiberg (SCCEIV), Ivo Kubjas (SCCEIV), Ahto Truu (Guardtime AS),

Peter Gazi (IOHK), Nikos Karagiannidis (IOHK)

Reviewers

Andres Ojamaa (Guardtime AS), Nikos Voutsinas (GUNET)

31 December 2018

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

Contents

1	Introduction	1
2	Requirements	3
2.1	Online Voting	3
2.2	Insurance	6
2.3	University Diplomas	8
2.4	Cardano Stake-based Ledger	10
2.5	Common Ground	12
3	Functional Interface Design	13
3.1	Online Voting	13
3.1.1	Actors, Components and Assumptions	14
3.1.2	Overall Architecture	15
3.2	Insurance	16
3.2.1	Data Exchange Model	16
3.2.2	High-level API Calls	17
3.3	University Diplomas	18
3.3.1	Overall Architecture	18
3.3.2	High-level API Calls	19
3.4	Cardano Stake-based Ledger	20
3.4.1	Overall Architecture	20
3.4.2	High-level API Calls	21
4	Conclusion	23

Chapter 1

Introduction

We present the requirements for all the PRIViLEDGE use cases, namely, online voting, insurance, university diplomas, and Cardano stake-based ledger. Our requirement analysis covers use case specific challenges, cryptographic technologies, and protocols. We also introduce a high-level interface design and provide an architectural overview for each use case.

Before we proceed with the requirements, a short overview of the interaction of Work Package 1 (WP1) with the other work packages in the project is in order. In particular, the goal of WP1 is to derive a set of requirements from the four use cases that will be fed to the other technical work packages. These requirements will investigate solutions, and guide development, while ongoing work will provide feedback to WP1 requirements throughout the project. At the end of PRIViLEDGE, one should be able to validate, through WP1, that the requirements have been met. For the sake of convenience, we summarize the technical work packages that WP1 will be interacting with.

WP2: Privacy-Enhancing Cryptography This work package will define a suite of advanced privacy-preserving cryptographic primitives that are useful in cryptographic protocols and in particular in the protocols required by the use cases of the project. Furthermore, WP2 will define the privacy-enhancing cryptographic notions that are needed for WP3 and the implementation of the use cases.

WP3: Cryptographic Protocols Work Package 3 will define a set of core protocols for building distributed ledgers. The protocols will use the cryptographic primitives defined in WP2, and must be able to support the requirements of the use cases defined in WP1. In particular, the protocols will cover:

- *Ledger protocols*: Consensus mechanisms, including mechanisms for generalised consensus, such as stake-based and flexible quorums. They will also cover ledger updates and authentication within the ledger infrastructure.
- *Secure multiparty computation*: Setting up a secure multiparty computation; support verifiable inputs for secure multiparty computations; running a secure multiparty computation inside the ledger; secure multiparty computations that hide the function being evaluated.
- *Post-quantum cryptographic protocols*: PRIViLEDGE will not ignore the implications of quantum computing and will investigate the use of post-quantum secure cryptographic protocols for specific operations.

WP4: Architecture and Deployment In the context of this work package, PRIViLEDGE will integrate the technologies investigated and developed during the project into a framework aligned with the use cases. This will involve:

- *A Generic Architecture* which will dictate how privacy-preserving cryptographic protocols and cryptographic ledger technologies can be integrated and applied together in different contexts.

D1.1 – Requirements & Interface Design

- *Toolkits* that will implement the specific technologies needed by the use cases, as identified by WP1. Such toolkits will include:
 1. An implementation of cryptographic protocols for anonymous authentication within a distributed ledger system.
 2. A toolkit for flexible consensus protocols.
 3. A toolkit implementing post-quantum secure protocols for distributed ledgers.
 4. A toolkit for zero-knowledge proofs for ledgers.
 5. A toolkit for ledger-oriented secure two/multi-party computation.
 6. A toolkit for privacy-preserving applications on top of ledger technology, in particular storing data on ledgers in a privacy-preserving way.

Chapter 2

Requirements

For each use case we discuss use case specific requirements, requirements related to the toolkits, and finally requirements that involve the ledger that is going to be used in the context of the particular use case. Then we discuss the common ground between the different use cases and identify similar requirements.

2.1 Online Voting

General Requirements

The ledger-related requirements of the online voting use case relate to the use of a ledger to ensure election integrity. In particular, the ledger is going to be used as a distributed bulletin board that in principle anyone may have access to. The distributed bulletin board should not be run solely by the election authorities as this would require election authorities to be trusted. To enable the involvement of external auditors, the underlying ledger technology must be packaged and available to facilitate the effortless set up of a participant in a tailor-made distributed bulletin board.

Note that, although the use of an existing general-purpose ledger infrastructure, such as Bitcoin or the Ethereum blockchain, would be a desirable property from an organizational point of view, it cannot provide the required security guarantees, because of the intrinsic performance issues pointed out by Heiberg et al. [3]. In order to be useful for voting, a ledger should accept authorized commitments immediately and unconditionally. If we look at existing proposals for blockchains to be used for voting, we do not see a comprehensive analysis of the conditions that need to be verified in order to thoroughly audit an election. In most occasions, we see a tendency for centralization, no guarantees of transaction acceptance, and performance limitations.

In the following, we focus on the private permissioned setting, while still using a ledger technology that has not been created solely for the purposes of online voting, but a general-purpose, existing technology under public scrutiny, that will be used for elections, alongside its other uses. At the time of this writing the decision for the suitable framework for the online voting ledger has not been finalized.

The ledger in the online voting use case is made available to external auditors. Through the ledger, they get access to any data committed to it by the vote collection service that is run by the online voting service provider nominated by the election organizers. This data is used after the tally to verify that the published voting result corresponds to the unaltered contents of the digital ballot box, where the votes were cast by the eligible voters.

Even though the ledger is permissioned, we foresee that anyone may apply to participate as an auditor. This means that the information published to the ledger must be sufficient to verify the integrity of the voting result, but has to be created in a voter privacy preserving manner. Specifically, we cannot publish digitally signed encrypted votes to the ledger directly as the computational assumptions ensuring the ballot secrecy may weaken over a longer period of time, creating a situation where a malicious auditor can determine the correspondence between voter identities and their respective choices.

We identify the following general requirements for implementing a universally available ledger for online voting:

- Universal verifiability of the correctness of the tally with respect to the data committed to the ledger.
- Individual verifiability of the commitment consistency with the voter intention.
- Everlasting privacy of voter choices with respect to external auditors.
- Computational privacy of voter identities with respect to external auditors.

It is highly desirable, albeit not absolutely mandatory, for the auditing to be compatible with the notion of re-voting—a voter can cast multiple ballots, which are all subject to commitment to the ledger, whereas only the last one of those ballots shall be taken into account.

It is highly desirable, albeit not absolutely mandatory, for the eligibility verification scheme to be updatable, in terms of modifying the electoral roll during the elections.

Toolkits Requirements

The online voting use case will require the following properties from the toolkits that will be implemented in WP4:

- **An implementation of cryptographic protocols for anonymous authentication within a distributed ledger system:** The voting use case currently has no requirements from this toolkit—the authentication and eligibility verification of the voters is not implemented on top of the ledger.
- **A toolkit for flexible consensus protocols:** The use case requires that it should be trivial to add and remove consensus network members. The consensus protocol should be trivially set up by the external parties, who may not have access to dedicated hardware. This means that the implementation should be able to connect through firewalls, be usable with unreliable connections, be resilient to unexpected halts, etc.
- **A toolkit implementing post-quantum secure protocols for distributed ledgers:** The use case considers quantum adversaries in two ways: preserving the authenticity of the ballots and preserving the privacy of the ballots. For the first case, as the elections are usually short-lived, there needs to be resistance against quantum adversaries for no more than a few weeks. However, as the ballots are private information, the ballot contents should stay private for the lifetime of the voter.
- **A toolkit for zero-knowledge proofs for ledgers:** The post-quantum algorithms in the toolkits should allow the use of zero-knowledge (ZK) proofs. The online voting use case requires the following ZK proofs:
 - proof of correct decryption
 - proof of plaintext knowledge
 - proof of plaintext content (i.e., the ciphertext contains an encryption of a value within an interval)
 - proof of plaintext aggregate (i.e., ciphertexts sum up to a value)
 - proof of commitment re-randomization correctness.
- **A toolkit for ledger-oriented secure two / multi-party computation:** The voting use case currently has no requirements from this toolkit.

- **A toolkit for privacy-preserving applications on top of ledger technology, in particular storing data on ledgers in a privacy-preserving way:** The use case will be using the distributed ledger for storing the commitments to votes and other public data related to the election immutably. For storing the votes, the latency and commitment time should be short, so that the voting process is not inhibited (more on this below).

Ledger Requirements

Considering the protocol, the voting use case sets the following requirements to Distributed Ledger Technologies (DLTs):

- **Consensus speed:** As the voting applications need to confirm to the voter that the ballot has been stored successfully, then the consensus should be obtained very quickly. The expected time target should be within a couple of seconds.
- **Block confirmation speed:** As part of individual verification, the verification application also verifies that the ballot has been committed to a block. As individual verification is usually performed straight after voting, we would expect the block to be committed within 15–20 seconds.
- **Scalability:** As the voting activity is rather predictable, we do not require that the service would be suddenly scalable during operation. We have seen peak voting rates up to almost hundred votes per second. To be able to use a single instance for multiple elections, the blockchain should allow storing up to few hundred to thousand votes per second. Additionally, for transparency purposes it must be possible to deploy members of the consensus network to election organizer or auditor premises. The deployment requires that the consensus network parameters (members, threshold) must be configurable on the go.
- **Confidentiality:** The confidentiality of voters and their ballots is obtained using public key encryption and pseudonymisation. Thus, the voting use case does not require additional confidentiality capabilities from the DLTs.
- **Identity services:** The online voting use case requires a digital signature scheme, paired with digital identities, that could assure the integrity of the signed data by a member of the set of eligible voters without revealing the actual identity of a particular member. Ideally this scheme would allow re-voting, multi-channel voting, and changes to electoral rolls. This requirement must be implemented independently of the ledger technology, so there are no additional requirements for identity service capabilities to the DLTs.
- **Cryptographic algorithms:** The online voting use case requires an asymmetric encryption scheme with homomorphic property and homomorphic perfectly hiding commitment scheme. The encryption scheme must allow for adding binary choices for tabulation. The commitment scheme must allow for adding the commitments to obtain the commitment for the aggregated ciphertext. Additionally, due to possible developments in cryptographic algorithms, the use case would also consider the usage of post-quantum secure protocols for distributed ledgers. This requirement must be implemented independently of the ledger technology.
- **Data storage:** The online voting use case requires that ballots are stored; the stored data will be a few kilobytes per ballot. Furthermore, the use case also requires the capability to retrieve the latest cast ballot by the voter.
- **APIs with clients and back-end applications:** The use case has the following API requirements:
 - *Lookup Endpoint:* Given an identifier obtained during data storage, it must be possible to obtain the stored value from the ledger. This interface must be available to all voters.

- *Search Endpoint*: Given a part of stored entry (in this particular use case, the election identifier stored during ballot storage), it must be possible to obtain all stored entries that contain the partial value. This interface must be available to election organizers and auditors.
- *Publish Endpoint*: Given a set of data and election identifier, it must be possible to publish this value on the ledger. This interface must be available to vote collection service, tabulation tool and election organizers. The data could be election configuration, auditor tools, votes, election result and corresponding proofs.

2.2 Insurance

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.¹

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.²

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.³

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.⁴ In practice, the ACO contracts are not pure outcome-based, but combine different payment models in various proportions.⁵

General 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.

¹https://en.wikipedia.org/wiki/Fee_for_service

²[https://en.wikipedia.org/wiki/Capitation_\(healthcare\)](https://en.wikipedia.org/wiki/Capitation_(healthcare))

³[https://en.wikipedia.org/wiki/Pay_for_performance_\(healthcare\)](https://en.wikipedia.org/wiki/Pay_for_performance_(healthcare))

⁴https://en.wikipedia.org/wiki/Accountable_care_organization

⁵<https://hcttf.org/acowhitepaper/>

- **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.

Toolkits Requirements

- **An implementation of cryptographic protocols for anonymous authentication within a distributed ledger system:** There might be a use for anonymous authentication in some markets where some services are available anonymously to all insured patients, but this is not a priority in the first prototype.
- **A toolkit for flexible consensus protocols:** A large market typically includes a number of care provisioning contracts running in parallel. As many of the participants may be involved in multiple contracts, the ability to execute several contracts with different subsets of parties on one ledger may have benefits.
- **A toolkit implementing post-quantum secure protocols for distributed ledgers:** As health records are extremely long-lived, the ledger-based commitments backing their integrity should be post-quantum secure.
- **A toolkit for zero-knowledge proofs for ledgers:** Getting trustworthy assurances on functions computed on otherwise “invisible” patient records is required feature for implementing outcome-based contracts. It remains to be seen whether zero-knowledge or multi-party computation provide better solutions, though.
- **A toolkit for ledger-oriented secure two / multi-party computation:** Same as above.
- **A toolkit for privacy-preserving applications on top of ledger technology, in particular storing data on ledgers in a privacy-preserving way:** Unlikely, as we don't plan on storing confidential information on the ledger.

Ledger Requirements

- **Consensus speed:** Not critical, as all actions are based on human input and most of the time different users in different organizations take turns updating the state of a patient's record.
- **Block confirmation speed:** Same as above.
- **Scalability:** A large care provider may have contracts with more than ten insurers. A large insurer may have contracts with more than 1000 providers. There could be more than 100 simultaneous contracts among various subsets of those parties.
- **Confidentiality:** The actual health records are highly confidential, but expected to be managed off-ledger.
- **Identity services:** Organizations should manage their own members and their access to internal records management systems. Parties writing to the ledger are identified by their pre-agreed public keys. Patients wishing to request transcripts of their own records are also identified by their public keys (most likely generated and managed on their mobile phones).
- **Cryptographic algorithms:** No definite requirements known at this time.
- **Data storage:** Most actual data is stored off-ledger. Only commitments of data updates and transfers are posted to the ledger.
- **APIs with clients and back-end applications:** Flexible. Access to the back-end ledger will be from custom integrations that will interface to record management systems used by care providers and insurers. Another custom integration will support patients reviewing their own records.

2.3 University Diplomas

In the context of the diplomas use case, there are three distinct entities, namely: *Issuers*, *HOLDERS*, and *Verifiers* (these concepts are in line with the World Wide Web Consortium’s “Verifiable Credentials Data Model” [2]). An Issuer is an organization (e.g., a university) that issues titles. Holders are persons that have titles and may want to present them to an interested entity. This entity can be an organization or a person that wishes to verify a title, i.e., a Verifier. To do so, the Verifier must contact the corresponding Issuer, who in turn, will provide the evidence needed.

General Requirements

We consider a protected blockchain setup: Only Issuers, Holders, and Verifiers may have write access to it. Reading can be allowed to several interested peers. Any reader can access the recorded transactions and verify them to the extent possible.

It is possible that in practice not all Holders and Verifiers may be able to write on the blockchain (say, because of low level of technological expertise). In that case, access to the blockchain may be mediated via gateways that act as proxies for Holders and Verifiers. If it turns out that such a scheme is necessary, for the sake of user friendliness and the adoption of the system by less technologically savvy users, care must be taken to ensure that the trust placed on gateways should not undermine the privacy and the security of the solution. Reading may be possible through multiple read peers of varied credibility.

- **Verifier Registration:** A registration service for the Verifiers may be required.
- **Proof Persistence:** The cryptographic proofs must be persistent. For that reason they will be stored in the blockchain (see below).
- **Auditing:** It must be possible to query the blockchain to produce lists of awards and proofs meeting specific criteria; the list will then be available for auditing purposes.
- **Commitment Granularity:** Although initially we want to commit whole diplomas, it is also desirable to be able to commit information at finer granularity (i.e., grades for specific courses).
- **Revocations:** As awarded diplomas are committed to the blockchain, they can never be removed from there. If an award is revoked for some reason, it must not be possible for the Issuer to provide certificates for it from the point of revocation onwards. One way to guarantee that is to require revocations to be entered into the blockchain as well.
- **Blockchain Entries:** The actual diplomas will not need to be entered to the blockchain, only encrypted hashes of them. Their individual hashes will be of small size. Apart from diplomas, cryptographic proofs will also be stored on the blockchain.
- **Transaction Volume:** Diplomas must be committed to the blockchain before the Issuer can produce certifications. If we want to enter past diplomas in the blockchain, then a large number of diplomas, possibly thousands or tens of thousands, must be entered en masse. A similar, although less severe situation, will appear if we want to enter all diplomas issued at a certain point in time (e.g., end of semester). This requires that either transactions can be entered in the blockchain in large volumes, or that we can work with suitable data structures incorporating hashes of individual hashes.
- **Open Verifiability:** Open verifiability means that the readers of the blockchain can verify a transaction and hold corresponding actors accountable.
- **Forensic Verifiability:** Forensic verifiability means that designated auditors can verify a transaction after retrieving necessary secrets from actors. In an actual deployment, the retrieval may be enforceable by technological means (e.g., a key escrow), or by legal means (imposed by legal procedures).

- **Open and Private Channels:** The blockchain itself is considered an open channel because a broad set of actors can read it. Open verifiability means that blockchain readers can collectively ensure the validity of transactions and uphold the accountability of transaction actors.

However, an open channel hinders both the private awarding of titles and the designation of verifier proofs. An open channel cannot hide secrets, while a private channel cannot be verified.

In the case of awarding, the issuer must commit to a secret value for the title and communicate that value to the holder. This requires an open channel for the commitment and a private channel for the transfer of the title.

In the case of proving, assuming that an issuer is reputable and hence considered to act honestly, a proof in the open channel that convinces the specified verifier should convince any other blockchain reader, because they all have the same information. Transferring the proof through a private channel between issuer and verifier supports designation but destroys (open) verifiability.

A trade-off we are considering is to embed and commit the private channels to the open blockchain as a series of encrypted transactions. This enables proofs going through private channels to allow for forensic verifiability.

Toolkits Requirements

In order to implement the diplomas use case, specific protocols will need to be implemented. To ensure re-usability beyond the confines of the project, the code will be structured in form of toolkits that will be usable by third party developers. In particular, the use case will relate to the development of the following toolkits:

- **Toolkit for privacy-preserving applications on top of ledger technology:** The toolkit may include both server-side and client-side libraries, to facilitate the development and the integration of front-end and back-end DLT applications.
- **University diploma record ledger:** The university diploma record ledger will function as an example and reference application of the technologies that will be available for use for certification and notarization services beyond the end of the project.

Ledger Requirements

- **Speed (of network consensus system):** Multiple transactions per second.
- **Speed (of block confirmation times, i.e., finality):** In minutes.
- **Scalability:** Not more than a few dozens of nodes.
- **Confidentiality:** Use of privacy-preserving cryptographic protocols.
- **Identity services:** Required, for all participants in a certification workflow. This requires: Issuers, Holders, Verifiers.
- **Cryptographic algorithms:** First designs require implementations of ElGamal, non-interactive (Fiat-Shamir) proof of a Decisional Diffie–Hellman tuple.
- **Data storage:** Diplomas will not be stored in the ledger; only encrypted hashes of data will be included in the ledger.
- **APIs with clients and back-end applications:** Interaction with the back-end DLT application will be through front-end applications. Initially, as web front-end will be developed. That will require implementation of the basic cryptographic protocols in JavaScript. Native applications in iOS and Android will be investigated.

2.4 Cardano Stake-based Ledger

The Cardano use case is qualitatively different from the other three use cases, as it concerns introducing a new feature to an existing blockchain, as opposed to using the blockchain concept in a new application. In particular, the requirements relate to the update system that can be put in place in Cardano, so that future versions of the software can be deployed in a running Cardano infrastructure securely and efficiently.

General Requirements

- **Bootstrapping:** The update mechanism should leverage the blockchain itself for the updates. So, decisions on what to update and when should be based on the same consensus mechanism.
- **Concurrency and Consistency:** Multiple concurrent requests for updates must be handled simultaneously, resolving conflicts. The adopted updates should be consistent, so that no contradictory updates are to be deployed at the same time.
- **Infrastructure Integrity:** Community splits over updates should be avoided.
- **No Lockout:** Participants who have been inactive when some update was deployed need to be prevented from getting locked out of the system.
- **Binary Linkage:** All update proposals must be linked to the certified binary code that will be deployed once the proposal is approved.

At this point, it is envisaged that updates can be effected using the side-chains mechanism [1]. This is a way for multiple blockchains to communicate with each other, so that one may react to events that happen on the other. A typical interaction between two blockchains is to transfer value from one blockchain to another; this functionality is called a *2-way peg*.

A sidechain system can be used to implement an update mechanism, by having the updates implemented in a sidechain. The value will be moved gradually from the original blockchain to the new one. If the sidechain is popular, the original blockchain will be abandoned, and we will have a single blockchain again. This scheme raises a couple of more specific requirements.

- **Sidechain Isolation:** Catastrophic events in the new chain should not impact the original chain. The only risks will be incurred to the assets that have been transferred to the new blockchain.
- **Independence from Consensus Mechanism:** The sidechain, along with the isolation requirement above, should be oblivious of the underlying consensus mechanism and should be applicable to both proof-of-stake and proof-of-work settings.

Toolkits Requirements

- **An implementation of cryptographic protocols for anonymous authentication within a distributed ledger system:** This use case may introduce requirements to this toolkit only in case additional privacy in the proposal-voting process is desired.
- **A toolkit for flexible consensus protocols:** The use case has no requirements from this toolkit, as it focuses on the Cardano blockchain.
- **A toolkit implementing post-quantum secure protocols for distributed ledgers:** Currently, the Cardano blockchain does not provide post-quantum security, and hence also the update system built on top of it cannot provide this feature. Nonetheless, this toolkit may turn out useful for a future update of the Cardano blockchain to provide post-quantum security, and this would be an excellent use of the developed update system.

- **A toolkit for zero-knowledge proofs for ledgers:** UC4 may introduce requirements to this toolkit only in case additional privacy in the proposal-voting process is desired.
- **A toolkit for ledger-oriented secure two / multi-party computation:** The use case has no requirements from this toolkit.
- **A toolkit for privacy-preserving applications on top of ledger technology, in particular storing data on ledgers in a privacy-preserving way:** The use case may introduce requirements to this toolkit only in case additional privacy in the proposal-voting process is desired.

Ledger Requirements

- **Consensus throughput:** The total size of the control messages of the update protocol that are going to be included into the blockchain have to be sufficiently limited so that they only exhaust a negligible fraction of the overall throughput of the blockchain, leaving most of it for “data” messages containing ledger transactions. In other words, the throughput of the underlying ledger needs to be sufficient to support the update mechanism without significant deterioration of its main service.
- **Consensus latency (confirmation times):** The blockchain update can be a relatively slow process compared to the practice-dictated necessary confirmation times for payment transactions, hence the update system will not introduce additional confirmation-time requirements on the blockchain.
- **Scalability:** Similar to the requirements on confirmation times discussed above.
- **Confidentiality:** The question of confidentiality is orthogonal to the update mechanism: while a basic update mechanism will identify the pseudonymous parties by their stake-controlling keys, additional requirements of full confidentiality in the proposal-voting procedure would imply additional confidentiality requirements for the whole stake-based blockchain.
- **Identity services:** Stakeholders participating in the update-proposal and update-voting process will be identified by their stake-controlling keys, no additional identity service is needed.
- **Cryptographic algorithms:** The sidechains implementation of the update mechanism will require new cryptographic primitives to securely certify the state of one chain to maintainers of another chain (connected via a pegging) that are not following it.
- **Data storage:** As in the case of throughput, the amount of data stored on the blockchain and serving as control messages to the update protocol should be negligible compared to the regular transactions.
- **APIs with clients and back-end applications:** The use case has the following API requirements:
 - *Update proposal submission endpoint:* A user must be able to submit an update proposal to the community. Appropriate API calls must be available that will allow for the uploading of the updated code to some content-addressable file system (or database), as well as the necessary update-metadata.
 - *Update proposal download and inspection endpoint:* An update proposal uniquely identified by a hash and stored to some content-addressable file system (or database) must be easy accessible for downloading and inspection (code and metadata). This will be necessary both at the voting stage, as well when a node decides to adopt the change and apply the update patch.
 - *Voting proposal endpoint:* A user must be able to vote for or against an update proposal, delegate her voting rights to some other user, as well as review voting statistics for a particular update proposal.
 - *Update proposal deployment endpoint:* A user must be able to provide through appropriate API calls a final confirmation for the application (deployment) of an adopted update proposal to her system. Moreover, appropriate API calls must exist for unapplying an updating, i.e., rolling back to a previous stable state.

- *Conflict resolution endpoint*: Appropriate API calls must enable the necessary resolution between conflicting updates by rolling back conflicting patches or applying required patches, in order to fulfill the dependency requirements of an update proposal.

2.5 Common Ground

We describe the requirements shared among the aforementioned use cases. Overall we have identified the following requirements:

- **Strong Confidentiality**: Blockchain data are by design to stay available for long terms, and indeed this is an important property that applications seek when using such technology. However, long term availability means that encrypted data on the blockchain may become exposed if the encryption algorithms are successfully attacked in the future. The insurance and voting use-cases are especially sensitive to this. The diplomas and the Cardano use cases may require strong confidentiality depending on future conditions. A post-quantum secure encryption scheme would greatly enhance security for all use-cases. Even without such a solution, effective methods have to be provided to be able to keep sensitive data off-chain while ensuring proper on-chain commitments.
- **Verifiability**: Using a blockchain brings a decentralized body of participants to guarantee important security and privacy properties of each use-case's transactions. The key tool that allows protect privacy while validating transactions is zero-knowledge proofs. Zero-knowledge proofs are certain to appear in the protocols for all use-cases.
- **Identity Services**: Managing the identities referenced in the different transactions is important in all the use cases, especially since under the new data protection regulation consent is required. The use cases need methods and tools to identify and authenticate entities, as well as manage the relevant cryptographic keys needed in the process.
- **APIs / Private Communication Channels**: These are needed to complete the functionalities of many use cases. The ledger itself and the protocols running on it are only part of the whole application logic for each use case. All use cases need external methods and systems in an architecture that uses the ledger as an independent component, as can be seen in the respective use case architecture figures in Chapter 3. In order to preserve privacy and ensure security, the ledger component must carefully provide appropriate APIs and anticipate private communication channels.

Chapter 3

Functional Interface Design

We describe the components of each use case at a high level and discuss how the different components interact with the corresponding ledger-based solution.

3.1 Online Voting

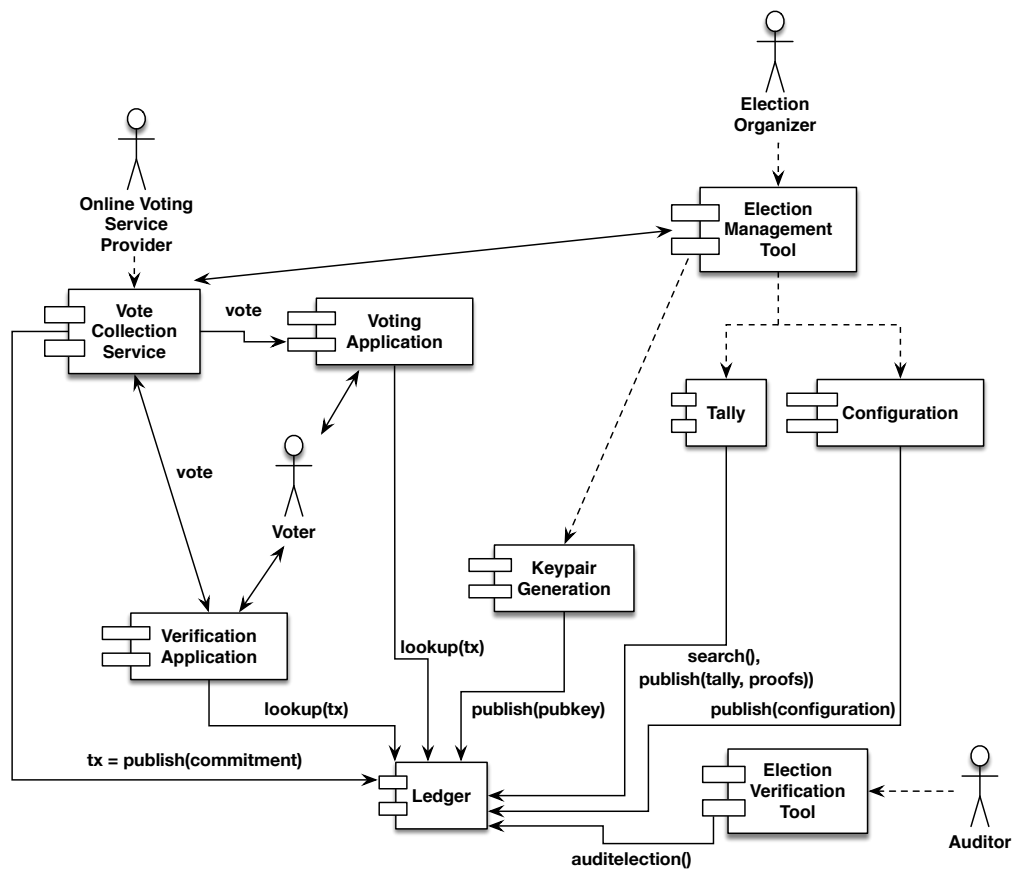


Figure 3.1: Components in Online Voting.

3.1.1 Actors, Components and Assumptions

The online voting use case deals with the following key actors:

- **Election Organizer:** It is the entity responsible for organizing the election and determining the election result based on the preferences of eligible voters. The Election Organizer nominates the set of candidates and the set of eligible voters.
- **Online Voting Service Provider:** The Election Organizer relies on the assistance of the Online Voting Service Provider that hosts the components necessary for running the election with online voting technology. The Election Organizer nominates the Online Voting Service Provider and tallies the election result based on the digital ballot box consisting of well-formed votes from eligible voters handed over by the Online Voting Service Provider.

The Election Organiser may only calculate the final tally after the end of the voting period that is previously agreed upon.

- **Voters:** They use the Voting Application provided to them by the Election Organizer and the Online Voting Service Provider to vote in the given election. For the sake of simplicity, we focus on the M of N (MofN) block votes, where a voter can make from 0 to M choices out of N candidates, which cover the vast majority of potential elections. Only eligible Voters are allowed to vote during the voting period. It may also be possible to vote multiple times, in which case only one vote shall be taken into account. The set of eligible voters may change during the voting period. All votes cast by eligible voters must remain valid even if the voter eligibility is revoked at later stage. It should be however possible to revoke a vote of a voter completely upon a request of Election Organizer.

Voters may also use the Verification Application to verify cast-as-intended and recorded-as-cast properties of their votes in a timely manner.

- **Auditors:** They can monitor and verify all phases of the election to ensure the compliance of the election with the election principles and the integrity of the election result. Hereby we focus on the Auditors who have access to the specification of the online voting protocol and the published data from a particular election. Based on this information, Auditors must be capable of verifying that the election result was calculated from the votes cast by the eligible Voters and no tampering of the individual votes, ballot-box, or the result has happened. In terms of online voting we speak of end-to-end verifiability and software independence. Auditors must not learn voters' identities nor for which candidates the particular voters voted.

This architecture assumes the existence of reliable, untappable electronic identities for voter eligibility verification and digital signature capabilities. These identities should ensure:

- privacy preserving eligibility verification;
- non-repudiation of the vote casting;
- correlation of the votes cast by the same person.

This protocol uses public-key cryptography for ensuring the ballot secrecy, more specifically, it relies on homomorphic tabulation of the voting result.

This proposal introduces distributed ledger technology as a part of the submission scheme, to facilitate the transparent storage of data for auditing the election.

The actors rely on following components to participate in the scheme:

- The Election Organizer uses the Election Management Tool for the election configuration, key pair generation and decryption of the homomorphic tally.

- The Online Voting Service Provider manages the Vote Collection Service to gather and store digitally signed encrypted votes from eligible voters.
- Voters use the Voting Application to select suitable candidate, compose the digitally signed encrypted vote and forward it to the Vote Collection Service for storage. The Voting Application is most likely provided by the Online Voting Service Provider, but is implemented according to a public specification so alternative implementations may exist.
- Voters may use the Verification Application to verify that the vote cast with the Voting Application was cast-as-intended and recorded-as-cast. The Verification Application is most likely provided by the Online Voting Service Provider, but is implemented according to a public specification so alternative implementations may exist.
- All votes stored by the Vote Collection Service are also registered to the distributed ledger, which serves as a basis of verification for Voters, Election Organizer and external Auditors.
- Auditors use the Election Verification Tool to analyze both data published by the Vote Collection Service and the Election Organizer to verify the integrity of the election result with respect to the distributed ledger.

It is assumed that the private key management with the Election Management Tool and the underlying homomorphic tallying scheme can be implemented in a threshold manner, so that the long term privacy of the voter relies on the quality of quorum.

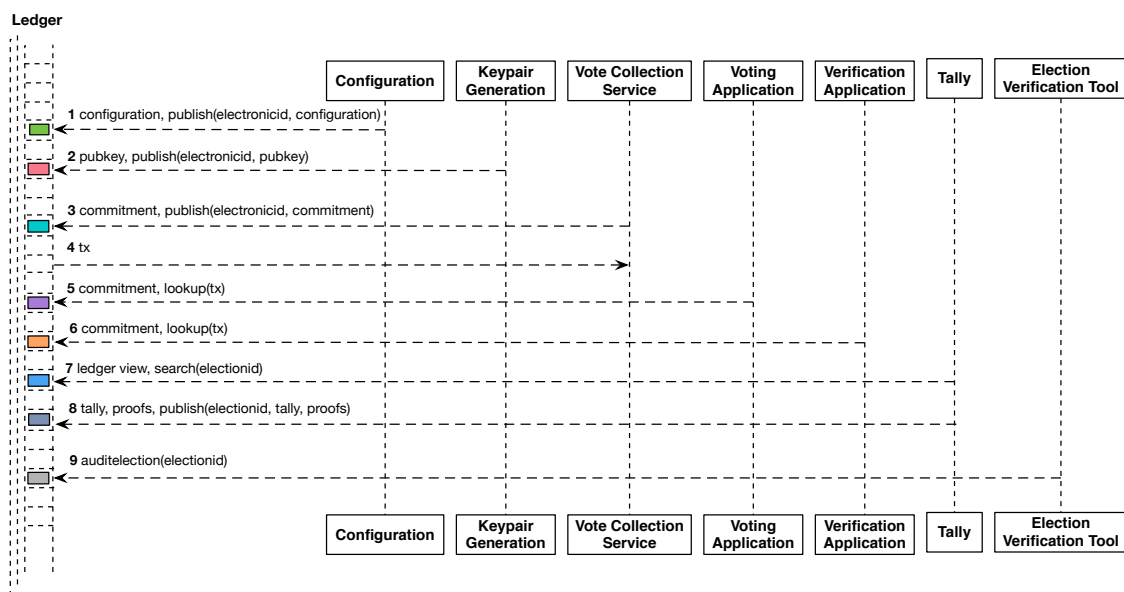


Figure 3.2: Interactions with the Ledger.

3.1.2 Overall Architecture

Figure 3.1 provides the overview of the overall architecture. The process of vote collection and tabulation must be transparent to any external Auditor, while respecting the voter privacy. The tabulation scheme is based on the homomorphic tallying, whereas the submission scheme must ensure everlasting privacy towards anybody who has access to audit data, but does not control the election private key.

Voters use the Voting Application to create commitment consistent encryption of the ballot. The Vote Collection Service extracts an information theoretically hiding commitment to the vote from the encryption and

publishes this to the distributed ledger together with the proof of membership to the set of eligible voters. The Vote Collection Service homomorphically aggregates all votes. The Election Committee decrypts the aggregate and publishes the result and opening to the aggregate commitment. Auditors verify proofs of decryption and that the opening published by the Electoral Committee corresponds to the aggregate commitment on the ledger.

This proposal requires the following high level API calls to the ledger:

- `publish(electionid, data)`
- `lookup(tx)`
- `search(electionid)`
- `auditelection(electionid)`

Figure 3.2 provides an overview of use of the ledger in the architecture. All election related data is associated with an ELECTIONID. Before the beginning of online voting period, the Election Organizer publishes CONFIGURATION (1) and PUBKEY (2). During the online voting period, the Vote Collection Service authenticates Voters, accepts each VOTE and publishes the corresponding COMMITMENT (3) to the ledger. The related TX (4) is provided to the Voting Application and captured by the Verification Application. Both applications use lookups (5), (6) to verify the successful storage of the vote. After the end of the voting period, a LEDGERVIEW is acquired by the Election Organizer (7) and used in the process of the tabulation of the actual votes received from the Vote Collection Service. The TALLY and the corresponding PROOFS are then published to the ledger (8). Once the TALLY is published, it is possible for the Auditor to verify all the data (9).

3.2 Insurance

3.2.1 Data Exchange Model

Both coordination of work within an Accountable Care Organization (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.

The overall process of providing health-care to a patient is as follows (see Figure 3.3):

- When a patient signs up for coverage with the insurance provider, the provider 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 3.4).

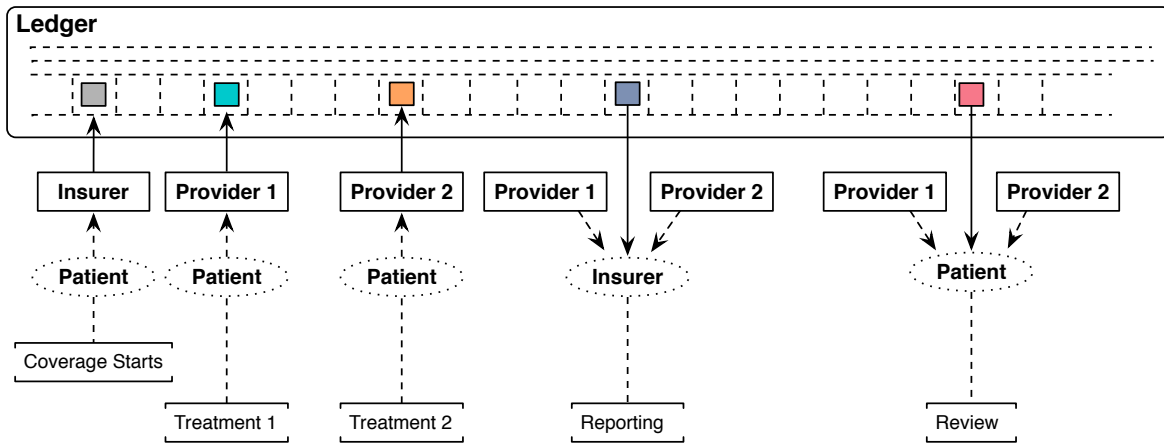


Figure 3.3: Healthcare provisioning process.

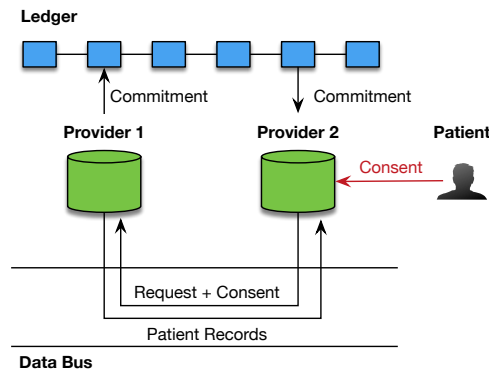


Figure 3.4: Referral data flow.

- 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 3.5).
- Optionally, patients may also review their own records to verify the providers no not claim visits or treatments on their behalf that have not happened.

3.2.2 High-level API Calls

- `commit(insurer, contract, patients)`: Post commitment of the list of patients covered by the given insurer under given contract; for some kinds of contracts, the providers can just ask the provider for the list; for others, the commitment must be such that the insurer can respond to a provider’s query with a proof showing whether a given patient is or is not included in the committed list;
- `fetch(insurer, contract)`: Retrieve the most recent commitment for the given contract;
- `commit(provider, records)`: Post commitment of a state of a provider’s patient record database; the commitment must be such that it would be possible for the provider to prove (a) to a patient or another

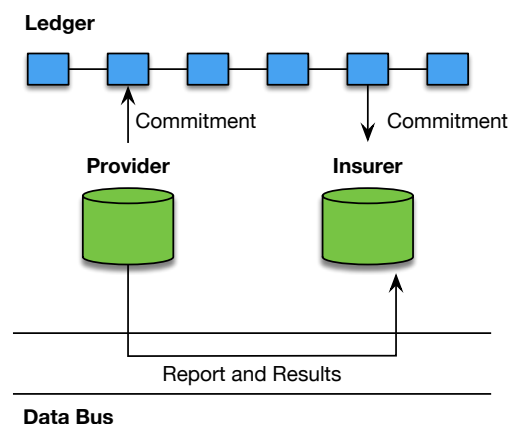


Figure 3.5: Reporting data flow.

provider that a given patient record is from the committed set; (b) to an insurer that the aggregate value given in a report matches the committed set; and (c) to an auditor that the new state is obtained from the previous one by only appending new events to patients' medical histories;

- `fetch(provider)`: Retrieve the most recent commitment from the given provider;
- `fetch(provider, commitment)`: Retrieve the last commitment from the given provider before the one given as the query parameter (for use by auditors).

3.3 University Diplomas

3.3.1 Overall Architecture

The architecture includes the following entities:

- **Application Service**: This service can be deployed in multiple instances so that each institution may use their own or one that they trust. It implements all the business logic, is responsible for end-user experience and key-management, and prepares the cryptographic protocol messages that will be sent to the Commit & Log Service (see below).
- **Commit & Log Service**: This is an abstraction over a blockchain network. There will be a stable API to record actions for issue title, request certification, certification proof, acknowledgment of proof. Different blockchain solutions may be employed at the back-end. Depending on the blockchain capabilities, part of the commit & log service protocol and other logic may be encoded as smart contracts or other advanced feature. The main value of this service is to provide an audit log of commitments for all participants so that they can be held accountable indefinitely. It is a general purpose service could be used for notarization services similar to diploma titles (e.g., real estate titles, any certifications, even some kind of self-sovereign identity solution).
- **National Authentication Service**: This will be the central point of synchronization of identities of persons across issuers and verifiers.
- **University student registry**: The university student registry can be connected to the Application Service to automate certification requests as per the policy that the institution decides.

- **Issuer end-user client:** The issuer end-user client allows the issuer to issue titles either on their own initiative, or in response to a holder request. It provides a view to the status of these processes, as well as provides a view to the activity that is automatically performed by the integration of the registry with the Application Service.
- **Holder end-user client:** The Holder end-user client helps holders manage their titles and their certification to verifiers. There are two main courses of action, requesting titles from issuers, and requesting certification to verifiers. All data are kept locally to the holder’s client to prevent breach of privacy.
- **Verifier end-user client:** Verifiers use their client to receive (or externally input) titles from holders and verify their authenticity. They can follow the progress in pending certifications as well as refer to past activity and check historic ones.

The clients will use a central portal to query all Diplomas service instances. The portal will present a unified view of the service and the process workflow to the end-user. At the same time, the certification data will only be available at the client side: the portal will not have access to unencrypted information.

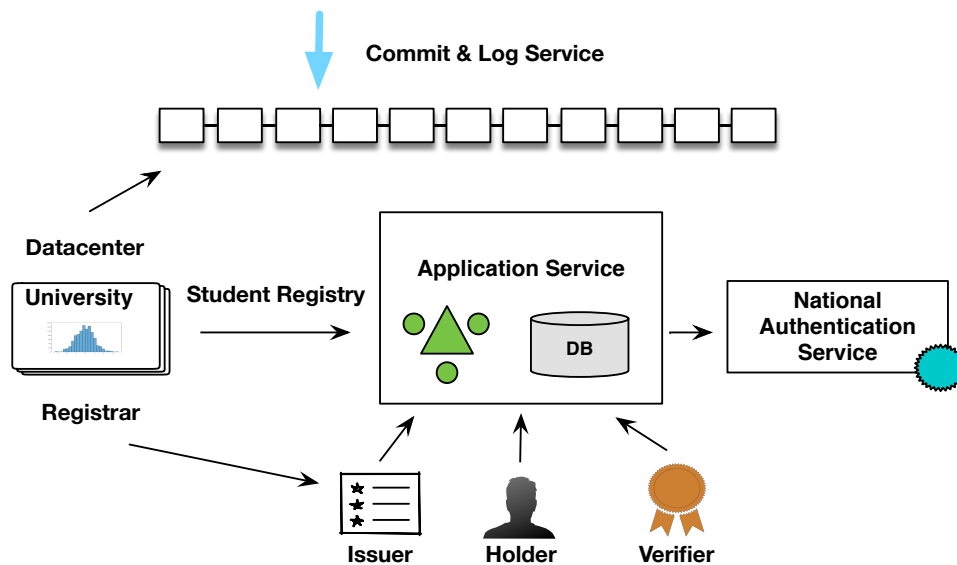


Figure 3.6: Overall Architecture of the University Diplomas Use Case.

3.3.2 High-level API Calls

Figure 3.7 illustrates the steps of the protocol. An Issuer issues a title (1) to a Holder in a private manner. However, the Issuer publicly commits to this act by recording a cryptographic commitment, which we call AWARD, to the blockchain.

Holders who wish to certify to specific Verifiers that they possess a title record a REQUEST (2) for the corresponding Issuer. In this way, they summon the Issuer to prove that they indeed have this title.

The Issuer identifies the request and initiates a PROOF (3) to convince the Verifier that there is indeed a title that corresponds to the Holder. A PROOF is derived from the initial AWARD.

Verifiers must record a challenge (CHALLENGE—4) to continue the proof private session. In turn, the Issuers respond (RESPONSE—5) to complete the proof. Verifiers publish an acknowledgment (ACK—6) to indicate that they were convinced and the Issuers have to admit (ADMIT—7) that the Verifiers have recovered the correct information.

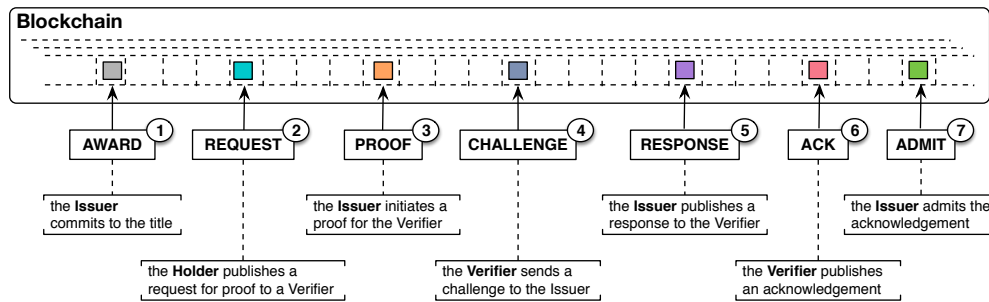


Figure 3.7: University Diplomas Protocol Basic Steps.

- `award(t)`
- `request(pkv, pkh, sawd)`
- `proof(sreq, pkv)`
- `challenge(sprf)`
- `response(sch, pki)`
- `ack(srsp)`
- `admit(sack)`

3.4 Cardano Stake-based Ledger

3.4.1 Overall Architecture

The architecture includes the following entities, as shown in Figure 3.8:

- **Proposals Subsystem:** This is an abstraction over a subsystem that accepts proposals from the various stakeholders of the Cardano system. Ideally, only participants with sufficient stake should have the right to submit a proposal for a change to the ledger. However, not all changes are the same thing and one can distinguish them along the following dimensions:
 - Whether a change directly impacts the ledger Consensus Rules and/or the validation logic of transactions and / or blocks.
 - Whether a change is a request for a new feature or a change request, or if it is a hot-fix for resolving some bug.
 - If it requires implementation, or if it is a change in the configuration of the ledger (e.g., the length of the epoch).

Depending on the “type” of the change according to the previous dimensions, as well as the various attributes (e.g., severity) that describe the change proposal in question, one can follow a different *update policy* approach. All proposals are categorized and inserted into persistent storage (“Proposal DB”) along with appropriate metadata.

- **Voting Subsystem:** This will be the main component for achieving a consensus on the priorities of the proposed changes. In order to enable a decentralized voting mechanism one could exploit a smart contract Cardano blockchain that could probably act as a sidechain to the main chain. For example an event on the main chain by a stakeholder could trigger a “voting event” on the smart contract sidechain. The ultimate output of the voting subsystem should be a prioritized list of proposed changes.

- **Changes Assignment Subsystem:** This will be the core component for managing the implementation of the changes. It receives as input a prioritized list of proposed changes. Note that apart from the priority, changes should be also accompanied by appropriate metadata, describing the type of the change, the severity (if it is a hot-fix), the part of the Cardano system that is affected etc. All these are recorded in a Change Requests DB. Then developers can choose which change they want to implement and an assignment task is created and recorded. After a change has been implemented, this has to be recorded in the Change Requests DB, in order the assignment task to be set to completed and a Pull Request event is generated, signaling that this change is ready to be deployed. The output of this module is a set of implemented changes (Pull Requests).
- **Core Cardano Update Subsystem:** This is the heart of the Cardano update system. It is the component where all update policies are materialized and the core ledger update mechanism is implemented. This module, at the most basic level, receives a set of implemented changes (Pull Requests) that must be deployed. Changes must be accompanied by appropriate metadata in order to be able to decide on the *update policy* that must be followed. We are planning to exploit the mechanism of sidechains, in order to achieve a smooth deployment of the changes into the ledger and at the same time fulfilling the requirements set at 2.4.
- **Databases:** It is natural that any update system requires some sort of persistent storage as well as database functionality. In the architecture that we have described, we have identified the need for two such databases. A “Proposal DB” where proposals for changes are recorded, along with appropriate metadata that will help us categorize the changes and then decide on the right update policy that must be followed; and a “Change Requests DB”, where implementation assignments are recorded, as well as the corresponding Pull Requests that trigger the deployment to the Cardano System.

The need of these databases in the Cardano update system architecture does not necessarily mean that we should follow a centralized approach in order to enable this functionality. Within the scope of this project we plan to investigate whether decentralized structures, such as smart contract sidechains, or other solutions, could be leveraged for the same purpose.

3.4.2 High-level API Calls

- `submitProposal (Proposal)`: A user must be able to submit an update proposal to the community for voting and acceptance.
- `downloadProposal (ProposalId)`: A user must be able to download locally an update proposal.
- `inspectProposal (ProposalId)`: In order for a user to vote for or against a proposal must first carefully inspect both the code as well as the metadata describing the particular update proposal.
- `voteForProposal (ProposalId, Vote)`: A user must be able to vote for or against an update proposal.
- `delegateVote (UserId)`: A user might need to delegate her voting right to some other user, who trusts for voting appropriately for update proposals.
- `revokeVoteDelegation (UserId)`: A vote delegation must be revocable.
- `applyPatch (ProposalId)`: User confirmation for update deployment on her local system.
- `unapplyPatch (ProposalId)`: Rollback of an update.
- `conflictResolution (ProposalId)`: Triggering of conflict resolution process. This might entail the rolling back of specific updates as well as the additional deployment of required updates.

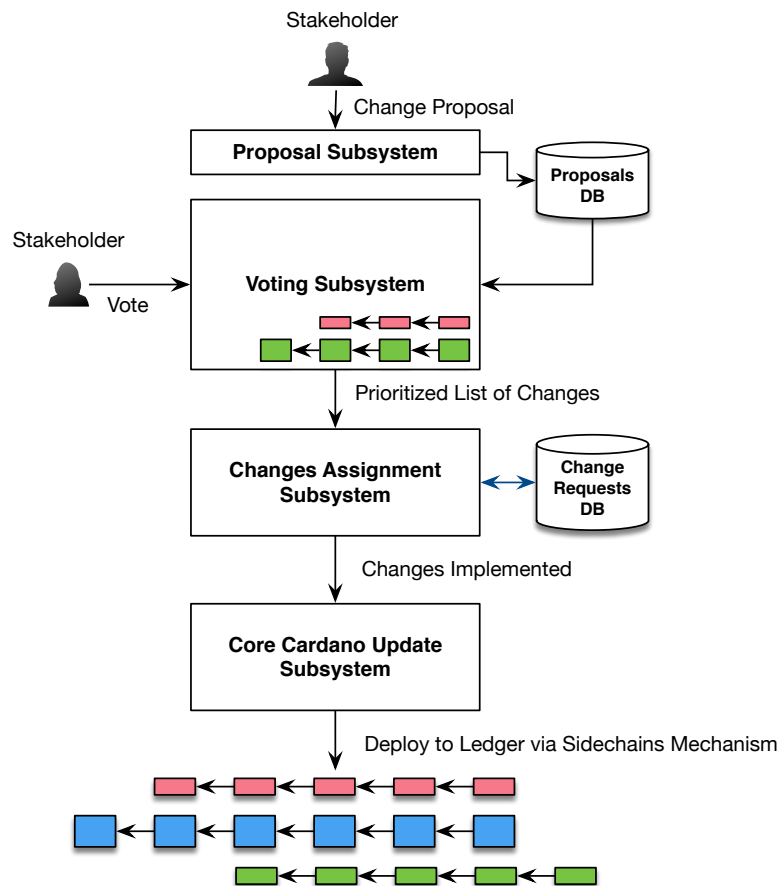


Figure 3.8: The Cardano Update System Overall Architecture.

Chapter 4

Conclusion

We have described a requirements analysis that covers all the use cases of the project. For each use case we provide the general requirements, toolkits-based requirements and the features required by the ledger to be used. We have also identified a number of common requirements including (1) strong confidentiality which in turn indicated a need for a robust post-quantum secure encryption scheme, (2) verifiability with zero-knowledge proofs, (3) identity services to manage the different transactions and (4) private communication channels to complete the functionalities for each use case (see Figure 4.1).

We also discussed the general architecture of each use case and how the various components may interact with the ledger at a high level. Furthermore, we provided potential actions that may take place in each use case. These actions usually take place in a specific order and they can potentially become the steps of a cryptographic protocol. Finally, our API calls description for each use case can guide the development of WP4 and lead to a generic architecture, which will dictate how privacy-preserving cryptographic protocols and cryptographic ledger technologies can be integrated and applied together in different contexts.

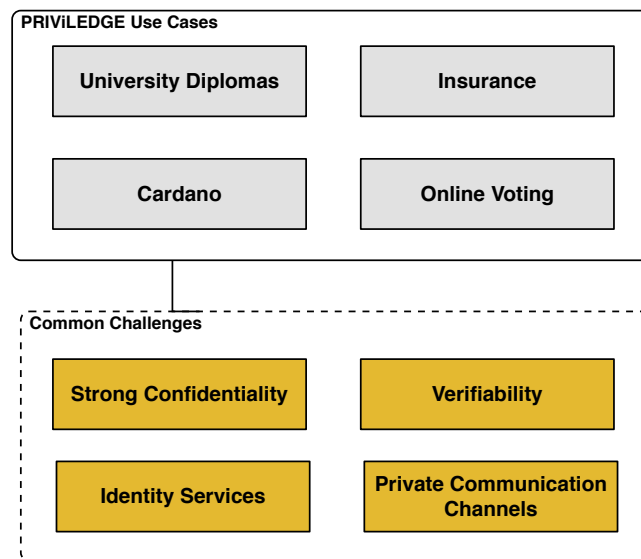


Figure 4.1: Common Requirements and Challenges Between Use Cases.

Bibliography

- [1] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. 2014. <http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains>.
- [2] World Wide Web Consortium. Verifiable credentials data model 1.0. Expressing verifiable information on the Web. <https://w3c.github.io/vc-data-model/#dfn-verifiable-credentials>. Accessed: 2018-02-02.
- [3] Sven Heiberg, Ivo Kubjas, Janno Siim, and Jan Willemson. On Trade-offs of Applying Block Chains for Electronic Voting Bulletin Boards. *E-Vote-ID 2018*, page 259, 2018.