LTO Network

Blockchain for Decentralized Workflows

www.lto.network

Abstract

Digitalization and automation of business processes offer great benefits in terms of productivity and cost reduction. However, organizations struggle to tap into these benefits for inter-organizational processes, partly due to a lack of trust. Bitcoin has proven how blockchain uses distribution and cryptography to provide a system that does not rely on trust.

LTO builds upon this foundation with a decentralized workflow engine, focussing on ad-hoc collaboration. Information is shared between parties using private blockchains (a new chain per process) and hashed on a public blockchain. This hybrid approach allows organizations to meet any data protection regulations and prevents scalability issues that are typically associated with blockchain projects.

Introduction

The digital revolution has resulted in many changes that focus on making our lives more efficient[1]. This wave of progress has primarily taken place in consumer-facing and internal business processes. When it comes to inter-organizational processes, we have to acknowledge that these changes are less drastic. Letters and faxes have largely been replaced by e-mail, and the typewriter has been replaced by a word processor but, beyond these superficial changes, the execution of the underlying processes has hardly changed.

The primary reason for automation being absent is the reluctance of corporations to rely on external systems that are operated by a counterparty[2], as distribution of information plays an important role in the outcome of a relationship[3]. Where there is an imbalance in natural power, one party may take control, forcing the others to use its centrally managed system. We observe this when dealing with the government and, to some extent, with corporations. In a situation where no single party can claim control, automation simply doesn't occur[4].

To solve issues regarding automation for inter-organizational processes, people have been experimenting with decentralized workflows for over two decades[5]. In these studies and experiments, a high level of trust and fair play is assumed, focusing primarily on solving technological challenges. In reality, this is a false assumption, as a lack of trust prevents successful pilots from going into production.

Another reason for the absence of automation is the correlation between efficiency and corruption[6]. Traditionally, large corporations and government bodies require a large number of people to execute a process. A fair amount of bureaucracy is also required to coordinate such processes. This increases the costs of bribery, reducing the incentive to automate. However, increasing efficiency negates this effect.

This paper will demonstrate how both problems can be solved by using blockchain, providing a solution where all parties can stand on equal footing.

Contents

Part I. Live Contracts

Business process modeling is a common strategy for any medium or large organization[7]. Creating a visual representation of a workflow process allows it to be analyzed, improved, and automated (figure 1). Unlike procedures that are written in a natural language or in a programming language, these models can be understood by both humans and computers.

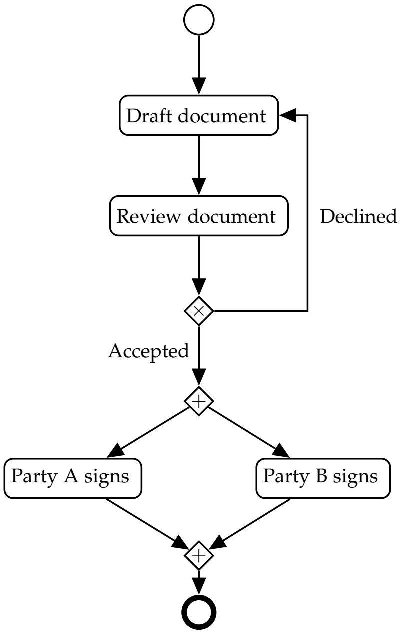


Fig. 1: *A BPMN diagram can be used to visualize workflows.*

For inter-organizational cooperation, modeling is not done merely to improve communication. The involved parties must specify the process, which will serve as a binding agreement[8]; on the LTO platform this is called a **Live Contract**.

The LTO platform creates an ad-hoc private blockchain for each Live Contract. Such a blockchain does not intend to serve as an immutable ledger but to ensure all parties have an up-to-date countersigned history of events and shared states.

1 Live vs Smart Contracts

Live Contracts have a similar goal as smart contracts as implemented on Ethereum[9]. Both define and solidify logic that can be applied in a trustless and verifiable way.

However, the philosophy behind these two types of digital contracts significantly differ. Ethereum describes smart contracts as cryptographic ″boxes″ that contain value, which are only unlocked if certain conditions are met[10].

Live Contracts do not directly hold value but describe how two or more parties should interact. Their intent is much closer to that of a traditional (paper) contract.

1.1 Ricardian Contracts

A Live Contract fits within the definition of a Ricardian Contract[1][11]. Most notably, it's easily readable by both people and programs. This is an emergent property obtained from the way a Live Contract is defined. There is neither a natural language version for legal purposes nor a coded version for programmed execution.

1.2 Enforcement

On-chain enforcement is poorly suited for many real world cases. Smart contracts rely on proactive enforcement, meaning breaching the agreement must be impossible or it must be possible for either side to drop out[12].

Take a non-disclosure agreement as example. The blockchain can't prevent a party from disclosing information, nor can it force a party to actively participate in resolving a breach. For such a contract to work as a self-enforcing agreement[13], it must hold the full penalty as deposit. This ensures that for every party it's in their own best interest to participate in a resolution.

Having to tie up large amounts of funds as deposits for penalty fees on arbitrary contracts is impractical for most organizations[14]. Additionally, the effectiveness of penalty interest and similar measures is limited to the value held by the smart contract.

Most business processes call for off-chain dispute resolution via an authoritative party. A Live Contract can facilitate the resolution of a dispute. This may include conflict negotiation, mediation, and even arbitration (by arbiter or judge).

Running a process on the LTO platform forms a verifiable history of events, reducing the amount of asymmetrical information. The distribution of information influences negotiations in case of a dispute[3] and influences the assessment conducted by an authoritative third party.

1.3 User interface

Ethereum provides an internal Turing-complete scripting language, which can be used by programmer to construct any smart contract or transaction type that can be mathematically defined[10]. This makes it very abstract, as the state contained within a running contract has no intrinsic meaning.

To interact with such contracts, a user interface must be created for a specific smart contract, or more precise the interface of such a contract[15]. Such interfaces can be standardized, like ERC-20[16] and ERC-721[16], to decouple the UI from the contract logic. The downside is that it also restrict the possibilities when designing a contract.

With Live Contracts, information does have an intrinsic meaning. While this restricts the use cases, it does enable the generation of an interface purely based on the data provided by the contract and its process. As a result, any workflow can be digitalized and executed on LTO without the need of creating a specific UI for each one.

1.4 Instructions and integrations

A Live Contract contains instructions intended for a specific person or node. A node may act upon a instruction or notify a user that it's expected to perform an action. Such an action can include getting information from the Internet via an HTTP API call.

The logic of smart contract, as implemented in Ethereum and Hyperledger, is only able to change the state of the blockchain. A smart contract requires an oracle to submit data from an external source to the blockchain[17].

[1] A Ricardian Contract can be defined as a single document that is a) a contract offered by an issuer to holders, b) a valuable right, held by holders and managed by the issuer, c) easily readable by people (like a contract on paper), d) readable by programs (parsable like a database), e) digitally signed, f) carries the keys and server information, and g) allied with a unique and secure identifier.

This oracle may act upon the state of the smart contract, however the logic of the oracle is not part of the contract. Such logic is part of a Live Contract, so it may be validated, and possibly disputed, by all parties involved.

2 Finite state machine

A Live Contract defines a workflow as a **Finite State Machine** (FSM)[18]. This makes it possible to visualize it as a flowchart (figure 2). That way, the workflow is understandable by both humans and computers.

2.1 Deterministic Finite State Machin

Any blockchain logic needs to be deterministic[19]. Where computer programs may require extra effort to comply with this, a Deterministic Finite State Machine (DFSM) is deterministic by definition.

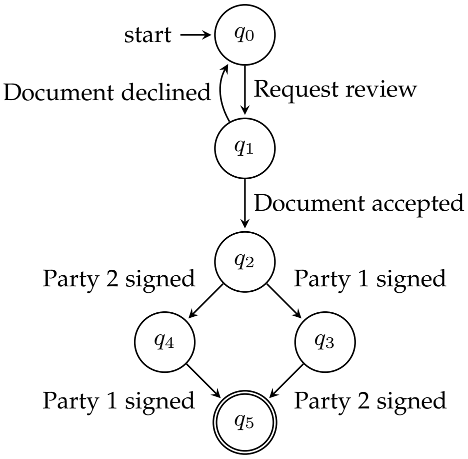


Fig. 2: *Example of a Finite State Machine visualized as a flowchart*

2.2 Extended Finite State Machine

Figure 2 illustrates how a problem arises when multiple actions are required to get to a state, but the order in which they occur is arbitrary. This *can* be modeled as a transition path for every possible order, as is done in Figure 2. However, with this approach, the number of states and state transitions will grow exponentially with the number of actions. This not only makes the visualization of the workflow less clear but also makes it more difficult and error-prone to define the workflow.

This is why instead of using a regular FSM, a Live Contract uses an **Extended Finite State Machine**[20] (EFSM),which allows conditional state transitions.

Figure 3 defines the same workflow as in Figure 2 using an EFSM.

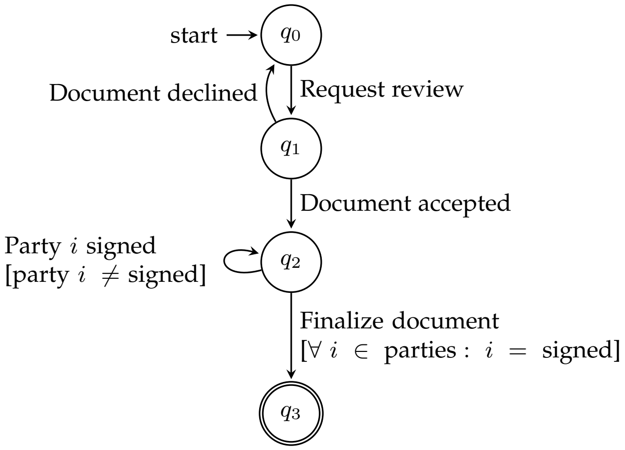


Fig. 3: *Example of an Extended Finite State Machine: Conditions in brackets have to be true for the transition to be valid*

2.3 Communicating finite state machines

Finite State Machines are limited to sequential behavior; they do not support concurrent processes. To represent workflows with concurrency, each sequence of parallel instructions may be represented as an individual FSM.

Communicating finite state machines (CEFSMs) allow modeling more complex processes by combining individual sequences of events.

The event chain (see Section 9) can function as a communication channel between two FSMs. When two processes are isolated by using different event chains, the communication channel is non-deterministic which, in turn, makes the whole system non-deterministic[21]. This can be overcome by acknowledging an event, as shown in Section 13.1.

2.4 Contract as automaton

A Finite State Machine can be applied as an agreement between the participants by formalizing obligations, permissions, and prohibitions parties impose on the other[22]. Contracts like financial agreements[23] and service contracts[24] can be fully digitized as an FSM.

However, representations as presented in these papers[23, 24], are not sufficient to be used as workflow, as they do not define the orchestration, communication, and choreography within a process. Though these factors can be incorporated, it causes the FSM to become exponentially more complex[25].

For practicality, an FSM will at best represent an incomplete contract. This doesn't necessarily have to be a problem, as these gaps can be filled with default rules[26]. The system does allow renegotiation of a Live Contract, either to resolve a particular situation or in general, as shown in Section 8.

Another thing to note is that not every action in a process constitutes as a binding factor. For example, in Figure 1, the acceptance of the text of a document does not constitute a binding agreement; this only occurs when the document is signed. To facilitate this distinction, actions can be categorized as either informative or performative [27].

3 Alternative modeling methodologies

Communicating EFSMs are commonly used to describe telecommunication systems and other real-time systems[28], but not business processes.

More common notations for workflows provide additional challenges when modeling decentralized inter-organizational processes.

3.1 Petri Nets

Petri nets[29] are a graphical representation of a system where multiple independent activities progress at the same time. The ability to model multiple concurrent activities differentiates Petri nets from FSMs. In an FSM, there is always a single "current" state that determines which action(s) can occur next. In Petri nets, there may be several states, and any one of them may evolve by changing the state of the Petri net. Some, or even all, of these states may evolve in parallel, causing several independent changes to the Petri net to occur at once[30].

Workflow nets (WF-nets), which may be used to describe business processes[31], are a subset of Petri nets. WF-nets can describe the whole process rather than only one sequence.

EFSMs use a global state to hold information. This information must be isolated per sequence. Failing to make the data immutable can be exploited, as shown in Section5.1. It isn't possible to use global information with Petri nets. Instead, information needs to flow through the workflow.

With the approach of CEFSMs, sequences are defined as individual processes. This makes applying access control to part of the overall process a trivial task. When representing the workflow as a whole, this can't be addressed in the same manner.

Petri nets have an interesting notation. Several studies[31] show that they can be used to represent a business workflow. Given it's possible to model CEFSMs as petri net, using WF-nets might be preferable to using individual FSMs.

Due to the similarity between Petri nets and FSMs, switching to WF-nets does not fundamentally change our solution, as explained in this paper.

3.2 BPMN

Business Process Model and Notation (BPMN) is the industry standard in business model processing and would be a likely candidate as modeling notation for LTO. However, there are a number of limitations that are particularly problematic for inter-organizational systems[32].

The Business Process Execution Language (BPEL), typically associated with BPMN, is a non-deterministic Turing complete language for web services[33]. This makes it ill-suited for automation on a blockchain.

A proposed alternative to this is to translate a BPMN model to a Petri net which, in turn, is translated to a smart contract[34].

While the translation to a (Turing complete) smart contract is unnecessary, the translation from BPMN to a Petri net (or CEFSM) might be interesting in order to support the current industry standard.

3.3 DEMO

"DEMO" is an acronym for "Design & Engineering Methodology for Organizations". This methodology for describing organizations and their business processes is based on "communicative action". It uses four models to create a holistic view, namely the Construction Model (CM), Process Model (PM), Action Model (AM), and Fact Model (FM)[35].

Rather than considering each step of a process individually, DEMO establishes a generalized workflow for every single performative transaction. In such a transaction, there are two roles, namely the initiator and the executor. The standard sequence proceeds as follows: the initiator makes a request, and the executor makes a promise. The executor then performs the actions and makes a statement about the result; the initiator either accepts this, completing the transaction, or rejects it[27].

Other alternative flows are also modeled, such as the executor declining the request, the initiator wanting to revoke the request, the executor being unable to fulfill the promise, etc. These alternative flows are often not, or only partially, modeled when using other modeling methodologies while, in practice, they are always present.

The Process Model (PM) combines these transactions to model a complete business process. The difference between this model and a workflow is that, in this model, everybody works in parallel as much as possible, specifying dependencies between transactions where needed. This design choice means DEMO models don't give a clear overview of where we are in a process compared to other modeling methods. Moreover, mutual exclusion of information (don't edit a document that's ready to be signed) is not immersive. Instead, it needs to be specified specifically.

DEMO might be a good way to make a high level model, which yields a workflow that can than be fine-tuned. This should result in more complete contracts, reducing the reliance on deviations (see Section 8.2).

4 Scenario

A workflow is defined as a data-object; *the scenario*. It consists of the following elements:

- q_x as a state with q_0 as the initial state,
- Q as the set of all possible states $Q = \{q_0, ..., q_{n-1}\}$,
- σ_x as an action,
- Σ as the set of all possible actions $\Sigma = \{\sigma_1, ..., \sigma_n\}$,
- δ as the transition function $\delta : Q \times \Sigma \rightarrow Q$,
- F as the set of final states with $F \subset Q \mid F = \emptyset$,
- $\bar{I}$ as all actor definitions,
- $\bar{A}$ as all asset definitions,
- D as the set of embedded data-objects.

4.1 States

States in set Q typically consist of the following:

- a title: a short title for the state,
- a set of actions with transaction as $\{(\overrightarrow{q_x}, \delta), ...\}$,
- a description: a long description for the state,
- a map or instructions for specific actors.

The state describes the actions that may be performed in this state and includes the state transition. This allows actions to be used in different states.

4.2 Actions

The scenario defines Σ, the set of all possible actions that can be performed in the workflow. Examples of such actions include filling out a form, reviewing a document, and performing an HTTP call. The action type and object properties are defined using JSON Schema.

An action defines which of the actors from set $\bar{I}$ may execute it and optionally additional constraints for execution. These constraints makes the scenario an *extended* FSM.

When an action is executed, it triggers a state transition. Actions can be categorized as manual actions, which require human interference to be executed, and system actions, which can be executed automatically by the system.

Optionally, actions can define instructions to update actors and assets using data from the response.

4.3 Actors

Set $\bar{I}$ defines all actors that can play a role in a process. Each actor is defined as an object using JSON Schema. Actor properties that are relevant to the process must be defined.

An actor in a scenario is only a static definition, which may be instantiated in a process.

4.4 Assets

$\bar{A}$ defines all assets that are available in a process. An asset is a variable data-object. Properties that are relevant to the process must be defined.

Note that the scenario only defines the structure of assets. Assets can only be instantiated within the process.

5 Data-objects

Apart from scenarios, other types of data-objects may be defined. All data-objects, including scenarios, use JSON schema as a type definition. Common examples of these are forms, documents, and templates.

Data-objects can be embedded in a process or linked and stored independently.

Linked objects are identified by the SHA256 hash of its JSON representation. To ensure JSON encoding the data always yields the same result, a deterministic method of JSON encoding is applied.

5.1 Immutability

Data-objects are immutable to the extent that, when a data-object is modified, it yields a new data-object. If this data-object is embedded in the process as an asset, the old object gets replaced with the modified one.

Notably, if a data-object is available in multiple processes, changes made the object in one process will not automatically propagate to other processes.

Failing to do so could lead to exploitable situations. In Figure 1, we demonstrated the process of negotiating and signing a document. It's clear that the document must not be allowed to be modified during the signing sequence.

5.2 Forms

A form definition uses a JSON Schema to define the data structure that should result from filling out a form. Optionally, an additional UI Schema can be used to specify how a corresponding field may be rendered and displayed.

There are several similar implementations[36–39] for this. Our aim is to work together with these projects to form a unified standard.

5.3 Documents

Digital workflows can largely eliminate the need for paper documents. However things like, legal compliance, backward compatibility, and company policy may still require the use of documents. By defining templates as a part of the Live Contract, natural language documents can be generated using data gathered by the process.

We recommend using the Open Document Format[40], which supports fields and conditional sections for creating templates.

5.4 Custom types

Any JSON schema that defines an object can be used as a data-object type. Custom types do entail the risk of a workflow not working properly, as other parties may participate via a node that doesn't support that type. Data with unknown types will be stored ”as is” and is unavailable outside of the context of the process.

6 Identities

An identity defines a person, team or organization within a Live Contract. An identity always contains the following information:

- identifier,
- node URI,
- custom information,
- sign keys,
- an encrypt key.

Identities are not the same as actors. An actor is an abstract role, such as a ”student”; however, an identity might be ”Bruce Willis” or ”Acme Corp”.

Sign-keys is a map with one or more public keys associated with an identity. The ”user” key belongs to the identity and can only be used by him/her to sign an action. The ”system” key is owned by the node that the identity uses and is used to sign automated actions. Other key types may be defined within a process.

The public encryption key can be used to encrypt data, which can only be decrypted by this identity.

6.1 Inviting identities

To add parties to a process, the scenario needs to define actions to add other identities. If the public keys are known within the process, the identity can be added directly.

When the public key is not known, the identity needs to be invited (figure 4). This can be done through any means deemed secure enough, including e-mail. The inviting system generates a one-time key and sends it to the invited identity. The invited party must replace this by its own secure ”user” and ”system” keys.

Before a new identity can fully participate in a process, additional authentication may be required. This can range from SMS verification to federated identity verification and even notary approval.

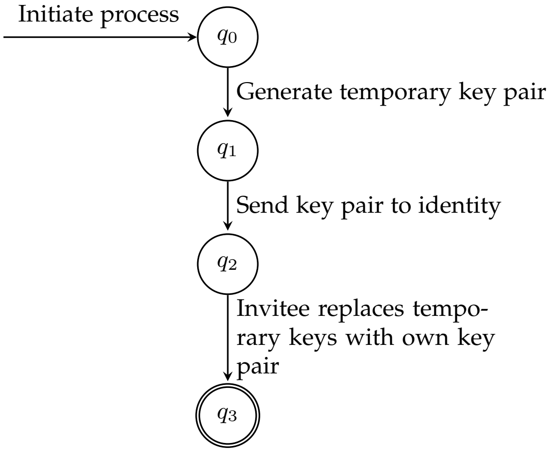

Initiate process
q_0
Generate temporary key pair
q_1
Send key pair to identity
q_2
Invitee replaces temporary keys with own key pair
q_3

Fig. 4: *Inviting an identity to join the process.*

6.2 Updating an identity

An identity is free to modify its own information, except for the identifier. This also allows a party to switch to another node. In cases where a user should not be allowed to switch nodes, it's up to the node of the identity to reject that change.

Removing an identity can be done by clearing the sign keys, encrypt key, and node URI.

Updating other identities is only possible if such an action is defined in the scenario and allowed in the current state.

7 Process

Where a scenario is the stateless definition of a workflow, the process is a stateful instantiation, consisting of the following:

- θ_x as response, where $f : q_x \mapsto \theta_x$
- Θ as an ordered list of all responses $\Theta = \{\theta_0, ...\theta_n\}$
- q_t as the current state
- I as set of all available actors
- A as set of created assets

7.1 Actions

Executing an action always yields a response. This response must be signed by the actor and submitted as a new event. Nodes independently determine the new state based on the current state and the executed action.

The scenario is defined as a deterministic FSM. However, this only concerns the state transitions and the projection. On systems such as Ethereum and Hyperledger, all logic must be deterministic, as it's executed by all nodes and needs to yield identical results on all systems[41].

With LTO, only a single node or a single actor executes an action, meaning actions do not need to be deterministic. As explained in section 1.4, oracles are not needed for actions like fetching data from an external source.

7.2 Manual actions

Applications built on the LTO platform must inform human actors about the actions they may perform in the current state. A human actor will sign his own response event before submitting it to his node, which will distribute it to all parties.

7.3 System actions

System actions do not require human interference but are executed by the node automatically. Therefore, instead of the human user, the node signs the response.

These actions are always performed by a single system and do not have to be deterministic. Other parties of the process validate the response and can reject it if needed. As there is no human interference involved in system actions, the actions are signed by the system itself instead of the actor.

It's also possible to schedule a system action to be executed at a later time. This is specified in the scenario. It allows for a timeout on a state or polling an external source at a predetermined frequency.

System actions are automatically executed and may yield an error when used incorrectly or otherwise fail. For these actions, not one but two state transitions must be defined. One for a successful execution and one in case of an error.

7.4 Sub-processes

A process based on an FSM can only be in one state at a time. Sub-processes allow a Live Contract to hold multiple states and make it possible for different procedures to be performed in parallel. While these processes share an event chain, the data of each process still remains isolated.

To facilitate sub-processes, a Live Contract may contain sub-scenarios that can be instantiated from the main scenario.

7.5 Projection

Apart from the FSM state, the process also contains other stateful data, such as assets and actors. The payload of every response can be used to update this data. The rules on how the payload updates the data are defined in the scenario. Updating the projection is deterministic and, therefore, applying a given set of responses against the scenario will always yield the same projection.

The projection can be used to set the parameters of an action as well as for the constraints defined for an action (see 4.2).

7.6 Data operators

Data operators may be used in the scenario for specifying how the projection affects the process. These operators are deterministic functions without side effects. They can be used for arithmetic or logical operations. The result of these operations may be stored in the projection and can be used as a base for, for example, state transitions.

7.7 Passive testing

A scenario that contains a loop consisting exclusively of system actions could result in an infinite loop, causing a massive number of transactions. When validating a scenario, we want to reject it if that have such a construct.

Determining if a program can run forever is known as the halting problem[42]. The problem has been proven unsolvable for Turing-complete machines; however, it can be solved for FSMs[43]. As an FSM has a finite number of transition paths, they can all be checked for loops.

Passive tests for EFSM models are complicated by the presence of unfeasible paths and is an open research problem[20, 44]. For simplicity reasons, we can assume any path through the FSM is feasible by ignoring the conditions. We accept that this may cause false positives.

8 Adaptive workflows

A scenario will model the most typical cases of a process. It's impossible to foresee all situations in advance and tedious to model every possible edge case. Taking a code-is-law approach would make the system rigid. Instead, Live Contracts supports three methods of resolving such issues.

- Comments
- Deviation
- Scenario updates

8.1 Comments

Comments are used to communicate with other identities. They can, for example, be used to resolve conflicts or conduct discussions outside of the process. Using comments instead of off-chain communication methods ensures that the conversations are logged on the blockchain. It also allows backtracking to check when in the procedure certain conversations took place.

Comments are not restricted to text messages. It is also possible to use images or documents to assist communication. Comments are not a part of the process, meaning that adding a comment does not trigger a state transition. Hence, it is always possible to conduct a discussion about subjects that were not predefined in the procedure.

8.2 Deviation

Any party may propose a deviation from the main flow by defining a partial scenario. This sub-flow must start from one of the states of an existing scenario and end in a state of that scenario. Deviation flows are only executed once, as they are no longer available when the process returns to an existing state.

All parties need to agree upon the deviation. Note that deviations might lead to forks that can only be resolved through manual conflict resolution.

A deviation can be used to resolve disputes. Any party may propose it to dispute the correctness of a previous event and present a solution on how to correct that.

A typical case of using deviations is making a payment arrangement. Organizations obviously don't want to make that option known within the agreement. Predefined sub-flows allow granting such arrangements, while keeping them under wraps.

8.3 Scenario updates

It may be required to change the scenario for a running process; for instance, when an agreement is updated or a new law is passed.

A party may provide a new scenario for a given process through a deviation flow. This flow moves the state out of the outdated scenario and into the new scenario.

Because updating the scenario is part of the sequence of events, it doesn't break the deterministic element of the solution.

9 Event chain

To determine the state of the FSM and the projection, we need to process the set of responses in the given order. Inserting or removing an event, changing the order of events, or modifying the payload may result in a radically different state.

In a centralized solution, the controlling party is responsible for data integrity. All parties rely on this party, as it represents a single source of truth. In a decentralized system, power and responsibility are shared by all parties.

To facilitate this, the event chain works like an ad-hoc private blockchain. Each response is wrapped in an event, which can be viewed as a block with a single action. These events form a hash chain, which is shared among the parties. The consensus algorithm ensures that the parties agree upon the sequence of events.

9.1 Permissionless private blockchain

Within a scenario, persons and organizations are defined as identity. An identity is free to choose with whichever node it wants to participate in the process.

There are no permissions required for a node to join the network. There are no permissions required to start a process and invite other participants. Data is only shared between participants on a process. As such, the event chain can be described as permisionless private blockchain.

9.2 Cryptographic signatures

To ensure nobody can falsify or forge events of others, each event is signed before being submitted using asymmetric cryptography. The signed event also serves as a receipt, allowing other parties to prove that the action has been executed by the signing identity.

The platform uses ED25519[45] signatures. These elliptic-curve signatures are widely used, well supported, and considered secure by institutions such as NIST[46] and ENISA[47]. Elliptic curve cryptography allows for faster single-signature verification and signing without losing security. It also reduces the needed size of both the keys and the signatures. Note that this method in itself doesn't grant complete security, as parties can still falsify or forge their own events. In other words, cryptographic signatures can't prove that an event did not occur.

9.3 Hash chain

Each event can be uniquely identified using its SHA-2 256bit hash. This industry standard algorithm guarantees fast computability and resistance to pre-image and second-preimage attacks as well as collisions[48]. It's the recommended cryptographic hashing algorithm by NIST[49].

Embedding the hash of the previous event in the hash of the next event creates a hash chain, which records the chronology of the events. When used in combination with cryptographic signatures, a hash chain provides an adequate measure of proofing that a specific sequence of events resulted in the current state[50].

10 Distribution

Rather than requiring parties to pull information from a central server or from each other, each party is responsible for pushing events to the system of all other parties involved.

Systems need to be always available so that events don't get lost. Decoupling and the use of a message queue reduce issues with temporary unavailability. In a typical case, all parties will connect to a node they trust, which will receive and process events for them. This node is a part of the larger system (see Section 19.1).

With the focus on organizations and governments, it's up to these organizations to run a node. Users connect to the node of their organization or a publicly available node of their choosing to participate in a process.

The CAP theorem implies that in the presence of a network partition, one has to choose between consistency and availability[51]. By using decoupling, we sacrifice consistency, enabling better availability. A node that is down or unreachable does not disrupt the process for the other participants and will simply sync up when it becomes available again.

10.1 Private chain

The event chain is a private chain that is only shared between the nodes chosen by the identities. Nodes are not aware of private chains that they are not a part of.

A node stores and facilitates many event chains simultaneously. Unlike side chains, event chains are completely isolated. Chains do not affect each other directly. This allows for horizontal scaling, given that the activity per event chain is reasonably low.

10.2 Genesis

Anybody may create a new event chain at will. The genesis block of this chain contains the identity of the user who's creating the process, and the subsequent block contains the scenario. As a part of the scenario, other identities will be invited to this private blockchain.

11 Consensus mechanism

LTO is a distributed system, where all parties can participate via their own node. Nodes distribute all events to their peers which, in turn, process these events. This means that there is a brief moment where the state of the process between the nodes differs. Eventual consistency[52] guarantees that eventually, given that there is no new event submitted, the state of the process on all nodes will be the same.

However, sometimes, new events are submitted before consistency is achieved. At this moment, it is possible that two or more nodes append an event to the event chain. During a Byzantine failure[53], all nodes believe their information is valid; however, the overall system is in an inconsistent state. In this state, nodes no longer accept new events from one another; they need to be able to come to a consensus, rather than halt.

Distributed applications use a different kind of consensus algorithm for this. In general, this is a case of *Byzantine fault tolerance*. Early Byzantine fault tolerance (BFT) methods do not scale well [54]. The invention of better scaling consensus algorithms, such as *proof of work (PoW)* [55], *proof of stake (PoS)* [56] and *proof of authorization (PoA)* [57], made it possible to create distributed networks with a large number of participants; this is also called distributed ledger technology.

While these consensus methods scale much better than traditional BFT methods, they need a relatively large number of participants (for PoW ¿ 1000) to be secure[58]. Traditional BFT methods rely on the fact that no more than $\lfloor\frac{n-1}{3}\rfloor$ participants are bad actors[59]. This means that with fewer than four participants, a single bad actor can influence the system and at least seven participants are required to be protected against two bad actors.

The event chain is a private blockchain with relatively few participants, often less than seven, meaning those algorithms are very vulnerable. Rather than trusting a majority vote, nodes consider their state correct unless proven otherwise.

11.1 Chance of a conflict

Event chains rely on optimistic concurrency control; a high number of conflicts would put a strain on the consensus algorithm, which can be relatively slow as it may have to wait on a block to be generated.

We define a distributed event chain as follows:

- Let N be the set of entities $\{n_1, n_2, n_3, \dots\}$ contributing to the event chain.
- Let C_n be the event chain, a sequence consisting of events $(e_1, e_2, e_3, \dots)$, belonging to entity n and let C be the set of all copies of the event chain $\{C_n | n \in N\}$.
- Let's define a conflict or branch as $\exists\ i, j \in N : i \neq j,\ C_{i_0} = C_{j_0}, C_i \not\subset C_j,\ C_j \not\subset C_i$.

For a conflict to occur by accident, two parties must add a block to their chain before they receive the update from the other chain.

Let's call the chance of somebody propagating an update to the chain $P(x)$. This chance depends on the number of blocks being added to the chain within a given time frame, the time it takes to propagate this block to the rest of the network, and the number of entities contributing to the chain within this time frame. Assuming everybody contributes equally to the network, this can be derived to Formula (1):

$$P(x) = \frac{f \cdot t}{n} \tag{1}$$

with:
f = Total amount of transactions / time frame
n = Total amount of active participants
t = Time it takes to propagate a block to the rest of the network

This chance can be used to calculate the probability of a conflict occurring. This probability is derived by subtracting the chance of not having a conflict from 1. When there is no conflict, it means either nobody has contributed to the chain at that moment, the chance of which is calculated using Formula (2),

$$(1 - P(x))^n \tag{2}$$

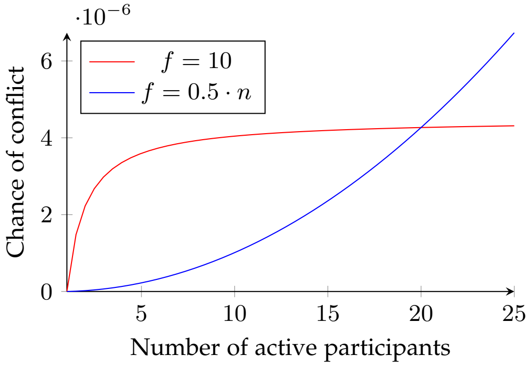
or only one node has contributed to the chain, the chance of which is calculated using Formula (3).

$$P(x) \cdot (1 - P(x))^{n-1} \cdot n. \tag{3}$$

Therefore, the chance of a conflict is calculated by (4).

$$P(c) = 1 - (1 - P(x))^n - P(x) \cdot (1 - P(x))^{n-1} \cdot n. \tag{4}$$

With a network delay of 1200ms, we see a chance of conflict of $< 0.0005\%$:



LTO works with a very small number of active participants on a single event chain; this reduces the chance of a conflict. With five or more active participants, the number is no longer relevant. With more than 10 participants, the chance of a conflict become more or less linear based on the network delay and transaction frequency.

If individually shared nothing event chains with decoupled message queues are used, the transaction frequency will be very low. This marginalizes the chance of a conflict.

11.2 Branch validation

A node can only prove the other party was aware of the chain up to the point of its last event. Any party is allowed to branch the chain after their last commit. If a party tries to branch the chain from a point before his last event, that branch is automatically discarded by all (other) parties, and the event is logged.

Before accepting the new events from the conflicting branch, they are validated like any received set of events. The event must be signed correctly by one of the identities and must be properly anchored. If the timestamp of the event differs more than 300 seconds from the timestamp of the anchor, the event may be rejected.

11.3 Order of anchoring

Nodes must anchor events on the global blockchain. The order of blocks if set by mining, the order of transactions on within mined blocks is also fixed.

This allows us to use the order of anchors on the global blockchain to determine the order of events. In case of a conflict, the block that was anchored first must be accepted. Consensus of the private event chain is reached via consensus on the public blockchain. On the public chain, consensus is reached by anonymous collaboration between a large number of participants using a variant on PoS.

11.4 Priority

It may be required to give an action or actor priority, so it's sequences first even if it was anchored last. It's possible to configure such priorities in the scenario.

Some event types, such as comments, naturally have a lower priority.

Using priorities enables front-running attacks, where an actor may respond to an event by creating a new branch that subsequently invalidates that event. Priorities should only be used in cases where this is no problem.

11.5 Unanchored events

When a block is received that has not been anchored yet, one may decide to accept the block anyway. This is, of course, if no conflict arises by accepting it. If the anchoring of the block has merely been delayed, accepting it would prevent unnecessary delays in the process itself. On the other hand, if the block is never anchored, no real problems arise. If everybody accepts the block, the process may continue as normal and the block may be anchored later.

11.6 Merging branches

When a fork happens, most blockchain applications will pick one chain to continue with and ignore everything that happened on the other branches. With a blockchain like Bitcoin, all transactions will be included in the mined blocks of each branch eventually.

On the event chain, the events themselves build up the hash chain. Picking one of the forks would lead to loss of information about an executed action. Instead, when a node becomes aware of another branch that has precedence over its own branch, it must base the events it has locally on top of the other chain. This is similar to a rebase action when using git[60].

11.7 Forks

Even though there is a guaranteed way to achieve consensus, a participating party might decide to ignore the other chains and keep the fork in place. For most blockchain applications, such as Ethereum[61], there is no reason to interfere with this, as the value comes from participating on the main chain only.

Live Contracts are a tool to digitize and partly automate existing processes. Even though the blockchain allows forks to exist, those processes usually do not. In the case of a fork, parties can start a secondary procedure to try to resolve the conflict manually.

12 Privacy

LTO is built for running processes between parties. Beyond these parties, no one needs to be aware of the process or even the collaboration.

Public blockchains allow anonymous accounts; however, these accounts function as pseudonyms. Any transaction may reveal the identity of an account, exposing the full transaction history. Smart contracts require the data to be public, as it needs to be available for every node. On consortium blockchains, participants are aware of each other.

ad-hoc private blockchains uniquely allow a random assortment of participants to collaborate without the need for approval or making information public. These blockchains can be completely erased when the process has been completed.

12.1 Linked data

Each party connects via its own node or a node it trusts. Each node has a private storage service where users can store data. Users have complete control over data stored here, similar to data stored on a service such as DropBox. They can remove their personal data at any time. The data is not shared or processed without a valid data processing agreement and explicit approval of the user.

When an action results in linked data, that data is not directly shared with other parties; only a hash is added to the blockchain. LTO prevents the hash from being used for anything other than verifying received data by putting it in an envelope together with a timestamp and some random data. The hash created to form the envelope will never occur on the blockchain more than once.

When an organization indicates that it wants to perform an action that requires linked data, the node of that organization will automatically make a request. The data owner's node checks if the specified action is valid and, thus, may indeed be performed by this actor in the current state.

12.2 GDPR

With the arrival of the new GDPR[62] (General Data Protection Regulation) in Europe, an argument has risen about the fact that a lot of blockchain applications are not GDPR compliant [63]. The two main reasons for this have been listed as follows:

- the fact that the blockchain's immutable nature is in conflict with the right to both amend and erase your data,
- the fact that there is no dedicated data controller, as it is a distributed environment.

Linked data means that the node chosen by the party to participate will act as the data controller for that user. All other parties always function as data processors. Data requests are automatically formatted to create a proper data processing agreement, which includes the purpose and time required for processing the data[64].

Ad-hoc blockchains allow the complete chain to be erased if required.

In short, LegalThings privacy features make the solution GDPR compliant without much additional effort.

12.3 Zero-knowledge proofs

A scenario may require one party to prove to another party that it knows a specific value. A zero-knowledge proof (zk-proof) is a method of doing so without conveying any information apart from the fact that it knows that value.

LTO supports zk-proofs through an interactive proof system. Two parties, the prover and the verifier, exchange messages with the aim of the prover convincing the verifier of its honesty (completeness) and of the verifier exposing a dishonest prover (soundness)[65].

A zk-proof for a Live Contract is always an action between two parties. There is no need for non-interactive zk-proofs such as zkSNARKS, which is still considered as experimental.

13 Common patterns

13.1 Chain interaction

Some processes may have to interact with other processes to be able to continue; for example, in case a process needs permission from another process to continue or to retrieve the result of a conflict resolution process. Requesting data from another process is done by following the pattern shown in Figure 5.

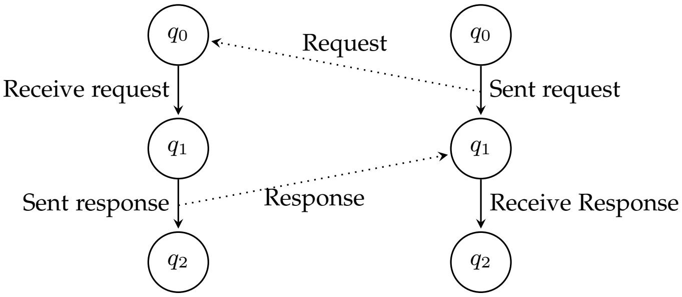


Fig. 5: *Pattern followed when two processes interact.*

13.2 Explicit synchronization

In some cases, it can be especially troublesome if an event is propagated too late. If this is the case, explicit synchronization can be built into the scenario. This requires all parties to acknowledge the current state before continuing the process. This is a pattern within the FSM, which intends to prevent parties from forking the chain of events prior to this event. In case of a dispute or conflict, such an acknowledgment can be used to decide on the branch that should be used to continue the process.

This kind of explicit synchronization only works if all parties acknowledge the current state. If some parties lack the incentive to continue or have the incentive to fork the process, another solution is required. In this case, the party that wants to continue the process announces its intention to all other parties. If any party that receives this announcement notices that the announced action is not valid on their chain, they can propagate this. This means that the chain has been forked and regular conflict resolution has to be applied. To remove the impact of network delays during this announcement, a set amount of time is waited before assuming that nobody has rejected the announced event.

Part II. Global blockchain

The LTO Global blockchain is a permissionless public blockchain, purposely built for verifying information. It exists to support Live Contracts and the private event chain. The global blockchain features anchoring and digital identities that are inter-operable across event chains, blockchains, and applications.

Notary transactions can be conducted on nearly any blockchain. However, on blockchains optimized for financial transactions or general logic, notary type of transactions are expensive and inefficient[66]. Furthermore, optimizations, such as pruning and sharding, can have a negative effect as relevant information is omitted[67, 68].

The global blockchain belongs to the Nxt family[69]. A unique characteristic of this blockchain is that transactions are based on a series of core transaction types that do not require any script processing or transaction input/output processing on the part of the network nodes. This reduces the blockchain's size, increases efficiency, and allows aggregation methods that are particularly beneficial for notary transactions.

Rather than starting from Nxt directly, we use a fork of the WAVES platform[70] as a basis. This platform has implemented a number of improvements, such as the NG protocol (section 15.6), which our network will benefit from. The existing transaction types focusing on digital assets, including colored coins, are removed or disabled and replaced by notary transaction types.

14 Centralized vs decentralized anchoring

Other solutions for blockchain anchoring use a centralized approach, where all hashes are collected for a period of time. From these transactions, a merkle tree is created, which is put in a single transaction on a third party blockchain like Bitcoin. Due to this, there is no direct feedback from the system, and it can take several hours before the final receipt can be collected[71] from the central service.

The LTO global blockchain is a decentralized solution, where every node oversees all transactions. When an anchoring transaction is broadcasted it's instantly visible, and after approximately 3 seconds it's pre-approved using NG (Section 15.6). The transaction is in a block within a minute.

Nodes can keep track of all anchored hashes or only of their own. If needed, they can create a receipt independently (section 16.1). There is no need for a centralized service.

15 Consensus algorithm

The global blockchain functions as a typical public blockchain. A node is selected as a generator to validate the transactions and forge a block. To determine the generator, we use the **Leased Proof of Importance** (LPoI) consensus algorithm. The generator is to be rewarded with the fees of the transactions in the forged block.

Proof of Importance is a variation of Proof of Stake (PoS), where the chance of being selected to forge a block is determined based on the number of tokens you hold and stake. With Proof of Importance, the chance increases based on the node's usage of the network [72, 73].

The token economy that emerges from this consensus algorithm is described in detail in the "LTO Token Economy" paper[74].

15.1 Leasing

NXT, Waves, and other blockchains in this family use Leased Proof of Stake[69, 75]. By leasing tokens, the token holder passes the right to forge a block to the selected node. These nodes can work like a mining pool, sharing the rewards proportionally among the leasers.

NXT style networks are susceptible to attack when an attacker controls at least 1/3 of all active balances[76]. This is an issue, as projects that have implemented LPoS algorithm tend to have a high level of centralization. The top two Nxt nodes control over 50% of the network[77]. With Waves, the top two nodes control over 1/3 the network, and the top five over 50%[78].

In Section 18 we will discuss the importance of a high amount of decentralization for this network. To prevent nodes with massive leased stakes, there is a limitation on leveraging leased tokens. Every node needs to own at least 10% of the tokens it stakes. Proof of Importance also counters this effect, as it puts nodes that are passive stakers at a disadvantage.

15.2 Raffle factor

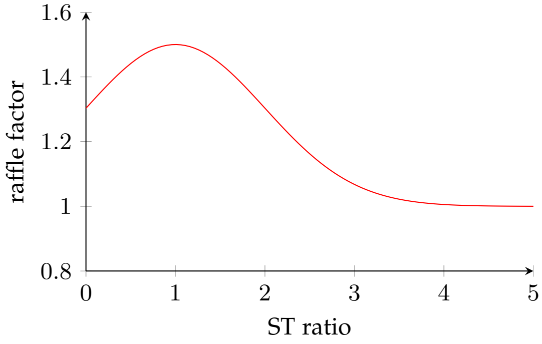
To calculate the usage of the network by the node, which influences the chance to forge a block, the balance of staking versus transactions (S/T-ratio) will be used.

$$\text{ST ratio} = \frac{\text{Staked tokens as \% of total}}{\text{Contributed transactions as \% of total}}$$

The S/T-ratio is related to a "raffle factor". The raffle factor is a mathematical formula that influences the chance of a node to be chosen to generate a new block. The more balanced the ST-ratio (closer to 1.0), the higher the Raffle factor. If the ST-ratio is unbalanced (a node does not contribute any transactions), the raffle factor will be 1.0.

$$\text{raffle factor } r = 1 + (0.5 \cdot e^{-0.5\cdot(\text{ST ratio}-1)^2})$$

This formula results in a large standard deviation of the bell-curve.

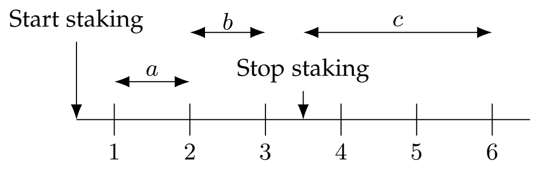


The maximum raffle factor is 1.5, which is halfway between the minimum and the absolute maximum of 2.0. To gain an additional X tokens from staking by inflating your importance, you need to spend and amount of $2 \cdot X$ on transaction fees. Importance inflation is explained in detail in section 18.1.

To fully understand the concept of the raffle factor and its effect on the token economy, please read the "LTO Token Economy" paper[74].

15.3 Forge probability

The chance of being selected to forge is $P(forge) = S \cdot r$. The contributed transactions T are calculated over time. When calculating $P(forge)$, S must be constant over the same period of time to prevent the possibility of abuse.



$a =$ Moment where P(forge) is calculated

$b =$ Moment where you forge a block

$c =$ Moment where tokens are still locked

Fig. 6: Timeline for staking tokens and forging blocks. Time is measured in number of summary blocks.

15.4 Fair PoS

The formula that decides which node is eligible to forge a new block is based on the Fair Proof of Stake algorithm[79], which has been created by Waves. This is an improvement on the original Nxt PoS algorithm, which overvalues higher stakes.

For a further understanding of the underlying algorithm, please read the "Fair Proof of Stake" paper[79].

To convert this algorithm from PoS to PoI, the formula applies the raffle factor to the staked balance which results in the effective balance as $b_i \cdot r$.

- T_i as block generation time for i-th account,
- X_n is the generation signature,
- r raffle factor,
- b_i percentage of staked vs total staked,
- Λ_n is the base target
- $T_{min} = 5$ is a constant for delay between blocks,
- $C_1 = 70$, a constant defining shape of delay distribution,
- $C_2 = 5E17$ is a constant to adjust base target.

$$T_i = T_{min} + C_1 \cdot log(1 - C_2 \cdot \frac{log(X_n/X_{max})}{r \cdot b_i \cdot \Lambda_n})$$

If you receive a new block before the allotted time is exceeded, this block must be added to the chain and a new time delay must be calculated. The previous T_i is no longer relevant. Each node calculates T_i itself; this information is not supplied by a generator. This means that there is no point of using an incorrect stake in the calculations.

15.5 Generator signature

The block hash is never considered while determining the time delay T_i. This hash is based on the contents of the block, which is determined by the node forging the block. If this hash played any part in determining who could forge the next, it would be easy to manipulate T_i. A node could create several different blocks and only broadcast the one that has the lowest time for itself.

To prevent this, only the generation signature is used. This signature is a secondary hash chain, using only the previous generation signature and the public key of the generator. With Nxt, this is completely deterministic, making it susceptible to be taken advantage of[76].

To combat this, as well as reduce the chance of forks, Fair PoS uses the generation signature used 100 blocks ago. Any change in balance on the node or in leases changes T_i for a given X_n. Nonetheless, any group controlling at least 1/3 of the tokens has a 30% advantage.

PoI requires the staked balance and raffle factor to be constant over a fixed period of blocks. This would make it even more vulnerable to such an attack.

As a solution, the generation is not a hash that can be publicly calculated. Instead, a node must hash the previous generation signature and sign that hash with its private key. This serves as a generation signature. In contrary to Nxt and Waves, a node can only calculate T_i for itself, making it impossible to determine the generators in advance.

15.6 NG protocol

The NG protocol was proposed to reduce the scalability issues on Bitcoin. While it was never implemented on Bitcoin, Waves NG has been active on the Waves main net since December 2017.

With NG, two types of blocks are generated, the micro block and the key block. The node that was previously elected may continue validating transactions, creating micro blocks on average every 3 seconds. When a new node is elected, it creates a key block from the outstanding micro blocks. Transactions in a micro block can be considered to be secure to some extent for low-risk transactions such as anchoring.

The reward of the transaction fees is split between the node that forged the micro block and the node that forged the key block as a 40% - 60% split. This split must always be in favor of the key block forger. Otherwise, there would be an incentive to disregard the already forged micro blocks and create new micro blocks yourself.

In a public stress test, Waves NG proved to be able to process up to 6000 Tx/min, with a peak of 17,000 Tx/min[80]. Potentially, the NG protocol could handle up to 1000 Tx/sec or 60,000 Tx/min.

NG reduces practical latency and is a key component for other optimizations.

16 TRANSACTION TYPES

The LTO global blockchain uses predefined transaction types. This allows for more compact blocks and removes the need for scripting. The list of transaction types may be extended in the future if needed. The types of transactions that are currently possible have been listed as follows:

- Anchor: used to verify transactions from the private blockchains,
- Issue a certificate: used to declare relationships between identities,
- Extend/revoke a certificate: used to extend or remove such a relationship,
- Transfer token: used to send tokens to another identity,
- Stake tokens: used to let a participant stake or lease tokens,
- Cancel staking: used to stop staking or leasing tokens,
- Set script transaction: used to configure smart accounts.

16.1 Anchoring

Anchoring is the method of taking a hash of a document or other data and storing it within a transaction on the blockchain. The goal here is to make it impossible for anyone, including the creator, to backdate or forwarddate his document[81].

Every event of the private event chain is anchored onto the global blockchain. Third-party applications may use the global chain to anchor documents for proof of existence. We estimate that 99% of all transactions on the global chain may be anchoring transactions. Considering that most transactions are for anchoring, aggregating these reduces the disk space required by the blockchain.

When forging a block, a node creates a Merkle tree[82] from the transactions in the order presented in the list. Only the Merkle root is added to the blockchain. As part of the validation process, each node recreates this Merkle tree.

Nodes are able to index every anchor hash. However, to reduce disk size, most nodes should opt to extract the Merkle path of its own anchor transactions. That path forms the receipt that can be stored with the original data (like the event).

16.2 Authentication and authorization

Challenge/response authentication methods, such as username and password verification, require a centralized system. In a fully decentralized system, we rely on cryptographic signatures to provide authentication. While information isn't shared between the event chains, parties can still be identified across chains, given the key pair used to sign.

In a broader sense, parties can sign any type of information this way. This is a similar use case as currently presented by PKI certificates. The reliance on central authorities to issue and revoke certificates has hindered the adoption of it as a replacement for challenge/response authentication.

With public blockchains, public/private key pairs can be created and used without the need for a central authority. A key pair forms a unique identity, which can be referenced to via an address that is derived from the hash of the public key.

16.3 Certificates

A certificate transaction allows every identity to convey information about another identity by referencing its address. Unlike tokens, granting and revoking an account is fully under the control of the issuer.

The certificate can be given a specific type, which is chosen by the party that issues the certificate. While not required, it's recommended that a certificate is acknowledged by the recipient account before being displayed to others.

16.4 Chain of trust

While a public address with a private key pair is a method of authentication, it doesn't provide a solution for authorization. Certificates can be used to specify relationships between identities.

This approach is similar to the web of trust (WoT). The WoT has a number of drawbacks over PKI, which we do not have on our platform.

On the blockchain, establishing and revoking a relationship or marking an identity as compromised is simple, instant, and irrevocable. Blockchain transactions are timestamped, which allows verifying the existence of relationships at a certain point in time.

Rather than simply establishing an identity, we establish a specific relationship. The transaction doesn't confirm or deny any other information about the identity, except the existence of the relationship, removing the need of physically meeting the other party.

In a given context, we only care about finding a chain of trust between two identities based on this relationship. This mimics the chain of trust as done with PKI validation but without the central authority. Rather than an absolute root certificate, the blockchain address of either our organization or an organization we do business with functions as trust root.

16.5 Smart accounts

By default, any transaction for an account must be signed using the private key associated with that account. Waves introduced the concept of smart accounts, allowing anyone to customize this logic[83].

To do so, this logic can be scripted using a non-Turing complete language. This script is only used to verify or deny a transaction for a specific account. It cannot trigger other transactions. Therefore, this type of smart contract does not prevent aggregation. Further limitations are placed on LTO smart accounts to ensure this.

LTO does not have data transactions, and other transactions are not accessible from the script.

Smart accounts can be used to create a multi-signature account by specifying alternative public keys that need to be used to sign a transactions. Rather than specifying these keys directly, it's specify that a transaction may be signed by anyone that holds a specific certificate.

There are other limitations that one might place on account as well. An account can be locked, only allow the transfer of tokens after a number of blocks, require a minimum number of tokens to remain on the account or only allow the transfer of tokens to specific accounts.

Using multi-signature is recommended when running a node. The account associated with the node typically holds a large number of tokens in order to stake and anchor. The private key to this account is known to the node. With multi-signature, obtaining that key won't give direct access to those tokens.

Anchoring transactions are unaffected by smart accounts. They always need to be signed with the private key of the account. This logic is in place to allow possible future optimization like horizontal scaling of the global blockchain node.

17 Summary blocks

Anchoring is a low-impact, non-disrupting method to bring an extra layer of security to the blockchain. We anticipate other applications that do not use Live Contracts to also use the anchoring feature.

One aspect of the blockchain that counteracts the scalability is the fact that the blockchain will keep growing[84]. The size puts certain requirements on the hardware to keep a copy. It also puts a burden on new nodes that have to play back the whole chain. To reduce the growing speed of the chain, summary blocks are used.

17.1 Key block size

Table 1 shows the structure of a key block on the blockchain. The global chain should be scalable up to $50\ million$ transactions per day. This is about five times the expected usage. The size of such a key block is determined by the block data and the transaction data (5).

$$Keyblock\ size = d + t \tag{5}$$

with:
$d =$ block data,
$t =$ transaction data.

Before calculating the expected block size, the following assumptions are made:

- 99.98% of the transactions are anchoring transactions (Table 5). This is the main use of the global blockchain,
- The other 0.02% are certificate transactions (Table 7),
- All other transactions are infrequent and can be neglected,
- All transactions are uniformly distributed over the blocks,
- On average, one key block per minute is generated,
- Every day, 1440 key blocks are created.

The block data is 277 bytes big (Table 1). Under the previously made assumptions, the size of the transaction data can be calculated using Equation (6).

$$Transaction\ data\ size = n \cdot (0.9998 \cdot a + 0.0002 \cdot c) \quad (6)$$

with:
$a =$ Size of an anchor transaction (Table 5),
$c =$ Size of an issue certificate transaction (table 7),
$n =$ Number of transactions per block.

This makes the total size of a key block about 3.8MB.

17.2 Growth without aggregation

With 3.8MB per block and 1440 blocks per day, the blockchain would grow 5.47GB per day / 2TB per year if it runs on full capacity continuously.

The expected usage is around 10 $million$ transactions, growing the blockchain 1.1GB per day. This results in about 3.65 $billion$ transaction or about 400GB per year.

For Bitcoin, with a total of 340 $million$ transactions[85], it takes about seven days to synchronize from genesis, depending on network and hardware speed.

With billions of transactions, doing this naively could mean waiting weeks or even months for the global blockchain to synchronize.

One of the goals of aggregating transactions is to require only 20 minutes of synchronization per year, again depending on network and hardware speed. With 365 summery blocks in a year, a node should be able to process a summery block within 3 seconds.

17.3 Segregated witness

Segregated witness is a strategy employed in Bitcoin to reduce the data in a block[86] by separating transactions into data that needs to be processed and data that is used to verify the transaction. This second part is called the witness data, containing signatures among other things.

Finality is a guarantee that blocks that are sufficiently deep will never be removed from the blockchain. Regardless of probabilistic finality or protocol finality, witness data is no longer useful if the block is guaranteed to not be reverted. Nodes are free to remove witness data for blocks that reached finality, saving disk space.

We've built on the logic of segregated witness, resulting int the concept of summary blocks.

17.4 Aggregation

These are special blocks that are created every 1440 blocks (about once a day). They contain aggregated values of all blocks since the previous summary block. When replaying the chain, only the summary blocks need to be applied to approach the current state. Then, only the blocks that were created after the last summary block have to be replayed. This decreases the replay time significantly.

The second to last summary block and all blocks before that are final. Nodes will not consider forks before that point, regardless of the longest chain. This means that only the transactions of the previous 1440 to 2880 key blocks have to be stored. Removing transaction data from the key blocks after it has been stored in a summary block reduces the key block size from 11.3MB to 277 bytes, making them negligible compared to the summary blocks.

17.5 Difference to pruning

At first glance, this approach appears to be similar to blockchain pruning, as you only maintain a limited set of transactions. The danger with pruning is that when a falsified state is introduced, it threatens the immutable nature.

The danger comes from distributing the state of the blockchain. With segregated witness, transactions are applied without signature validation, relying on the concept of finality. However, every node still needs to apply all transactions from genesis to calculate the current state.

The actual transaction data is not used to calculate the block's signature but rather stored as an attachment next to the block (Table 2). As events without transactions, key blocks are part of the blockchain and can't be ignored. If transactions are applied without validation, aggregating them holds little risk.

17.6 Summary block size

Summary blocks contain all information about non-anchor transactions and an aggregated version of all the transaction fees and other token transfers. They are rather larger blocks, especially compared to the key blocks.

To reduce the amount of memory used, the transaction fees and token transfer transactions get reduced to a balance change per participant (Table 4). The summary also contains non-aggregatable transactions, like certificate, staking and script transactions.

To calculate the expected size of a summary block, the following assumptions are made:

- There is a total of 200000 participants,
- Every day, one summary block is created,
- Based on the assumptions in the previous section, we can assume that the balance change summary is the only significant part of a summary block.

Using these assumptions, the size of a summary block can be calculated. When Equation (7) is used to calculate the size, the result is about 10.3MB.

$$Summary\ block\ size = Transaction\ summary = p \cdot e \quad (7)$$

$p =$ Amount of participants in the last 1500 blocks,
$e =$ Size of a balance change summary entry (table 4).

17.7 Total size

The total size of the blockchain consists of two parts, a static part and a growing part. The static part consists of the last 1000 key blocks . Those will still contain the attachment data (5).

$$Static\ part = n \cdot k \quad (8)$$

$n =$ Amount of keyblocks stored with transaction data,
$k =$ Size of a keyblock (5).

When Equation 8 is used to calculate the size of the probability, it demonstrates that the total size is about 11.3GB. As only the last 1000 blocks are stored completely, including transactions, this size may differ slightly but won't grow noticeably.

The growing part consists of the summary blocks (7) and what is left of the key blocks after the transaction data is removed. We'll define the size as the growth per year using Equation (9).

$$Growing\ part = n \cdot k + m \cdot s \tag{9}$$

with:
$n =$ Amount of keyblocks per year,
$m =$ Amount of summaryblocks per year,
$k =$ Size of a keyblock without transaction data,
$s =$ Size of a summaryblock (7).

Following the previously made assumptions, this equation demonstrates that the blockchain grows about 3.7GB per year.

17.8 History nodes

Nodes are not required to delete old transactions. By maintaining all transactions, history nodes can prove the correctness of the blockchain when needed. History nodes are unable to pass a falsified history, as the blocks of the history node needs to match those of other nodes. The network could rely on a relatively small number of history nodes.

There is no on-chain benefit of running a history node. It doesn't increase the chance of forging new blocks. Running such a node must be done out of community interest or secondary income.

18 Network vulnerability

18.1 Importance inflation

A particular worry with PoI is the inflation of importance through dummy transactions. We can calculate the profit/loss from spam transactions as a formula of the maximum raffle factor, which has been given below:

- Raffle factor; r,
- Percentage of staked tokens; b_i,
- Cost of a transaction; c,
- Total transactions on network; n,
- Spam transactions; τ,
- Rewards; p,
- Profit/loss from spam; $\Delta p = p_{r_{max}} - p_{r=1}$.

$$p = (r \cdot b_i \cdot n \cdot c) - (\tau \cdot c) \tag{10}$$

$$r = 1, \tau = 0 \rightarrow p = b_i \cdot n \cdot c \tag{11}$$

$$r = r_{max}, \tau = b_i \cdot n \rightarrow (r_{max} - 1) \cdot b_i \cdot n \cdot c \tag{12}$$

This gives

$$\Delta p = ((r_{max} - 2) \cdot b_i \cdot n \cdot c) \tag{13}$$

Given

$$b_i > 0, n > 0, c > 0, \Delta p < 0 \rightarrow (r_{max} - 2) < 0 \tag{14}$$

$$r_{max} < 2 \tag{15}$$

Equation 15 proves that it's impossible to gain directly from spam transactions with a maximum raffle factor of less than two.

A raffle factor close to two would make spam transactions nearly free. Increasing the importance of the network for little to no costs is undesirable, as it could aid an attacker trying to undermine the network with a 51% attack. The maximum raffle factor of 1.5 ensures a high cost of inflating importance.

18.2 Nothing at stake

The nothing at stake principle is the assumption that nodes will continue to build on any forks, rather than picking the longest chain, as there is no downside in doing so[87]. If all nodes would display such ill behavior, an attacker would only need a small percentage of tokens to force the network to switch to the other chain; a 1% attack.

Such a situation is referred to as a Tragedy Of The Commons[88]. All parties seek to individually gain by abusing the system. But, if everybody is doing it, nobody is benefiting from it. Instead, it only leads to undermining the network, causing a decrease in the value of the underlying token. https://v2.overleaf.com/project/5b95e00fe0915f6ac7833767 Waves Fair PoS has made it a lot more difficult for an unpublished orphaned branch to catch up with the main branch[79]. The possibility of profiting from such behavior with bad actors that only hold a small percentage of tokens is negligible.

A bad actor would need to create and maintain an altered version of the node. The costs, combined with the knowledge that there is little chance of benefiting from it, should be enough to disincentivize this behavior[89].

18.3 LPoS centralization

Projects that have implemented the LPoS algorithm tend to have a high level of centralization.

This effect can be explained by the reward per token. A professional setup with a near 100% uptime will not miss a forging opportunity, yielding more rewards with a set number of tokens being staked. This draws token holders to lease to these nodes and has a reinforcing effect, as reduced overhead allows for higher payouts to lessors.

Limiting the number of tokens per node is a flawed method, creating an opportunity for a Sybil attack[90]. In a permissionless reputation system, a single node can advert itself as multiple pseudonymous identities, circumventing this limitation.

Because of the nature of our solution, the majority of transactions will be signed by a key pair associated with a node. The network will also consist of a relatively large number of nodes. This has no effect on PoS; however, on PoI, it gives the users of the platform an advantage over non-user token holders.

An additional measure is the limitation on leveraging leased tokens; every node needs to own at least 10% of the tokens it stakes.

18.4 Denial of service attack

Due to limited scalability, it's fairly easy to overload any public blockchain with too many transactions. Transaction fees are the primary defense against these sorts of attacks, but transactions still need to be verified to conclude that there are insufficient funds. Moreover, with a decent amount of funds, it's possible to overload the network with spam transactions, as seen with Ethereum in July 2018.

Nodes have the option to automatically increase the transactions fees in case of such an attachment. Ideally, nodes gain as much from staking as they spend on transactions. As the spam tokens increase the rewards of transaction fees, this is used to automatically counter the attack.

18.5 SHA-2 vulnerability

In 2017, SHA-1 was proven to be vulnerable when the Google research lab managed to find a collision where two different documents resulted in the same hash[91]. If SHA-2 256bit is similarly vulnerable, this could be devastating for anchoring based on a Merkle tree.

If a collision is found, one can claim that the colliding document has been notarized. Even worse, given a Merkle root and a random hash, one might be able to generate a valid Merkle path. This would allow a hacker to verify any document. However, this would still not be an easy task, as, for every branch in the tree, two hashes need to be combined that are exactly 32 bytes long.

While bruteforcing a specific SHA-1 would still require too many computations to achieve in a lifetime, the birthday paradox results in much fewer computations being required to find a collision. The birthday paradox is also applicable to the LTO public chain, as there are many Merkle roots, each with a maximum number of Merkle paths.

To overcome this, verification might fall back on history nodes in case the verification is disputed. However, this reduces the overall use of anchoring nodes.

Instead, a secondary Merkle tree might be added where the SHA-2 hash is hashed with another algorithm such as SHA-3 or Blake2. Simply double hashing isn't useful when it's possible to falsify a Merkle, as only a vulnerability in the outer algorithm is required for an exploit. However, if there are two Merkle trees, both algorithms would need to be broken. However, even then, a collision needs to be found for both trees of a single block, removing the advantage of the birthday paradox.

Part III. Platform

19 Architecture

19.1 Micro architecture

The LTO Node is developed using the microservices architecture pattern. This means that all the functionalities of the node are split into microservices, with each service responsible for only a small part of the entire node. There are several advantages of this pattern, which have been listed as follows:

- Failure isolation: If a service fails, it won't necessarily interfere with other services.
- Scalability: All the services within the node are decoupled and can, therefore, run on different machines. This makes it really suitable for horizontal scaling. The scaling is automated, as explained in the description in Section 24.
- Flexibility: Certain functionalities flourish better in certain programming languages. Each service can be developed in a different programming language.
- Code quality: By splitting the node into small and well-defined modules, it becomes easier for developers to read and review. This leads to better code quality.

The microservices are grouped in a docker container. All these containers are run using the Kubernetes container orchestration platform; this will be described in Section 24. Each microservice is designed to run independently. This means that it has no shared dependencies, so each container has it's own database or event queue. Microservices are also designed to operate statelessly so that they are easily scalable.

19.2 Application layers and services

As described in Section 19.1, the node is split into several services. These services are grouped into four different layers. The node consists of the following layers:

- UI Layer: This consists of UI applications that interact with the application layer.
- Application layer: This contains all the services that handle actions that are triggered by events from the event chain,
- Private chain layer: This takes care of the decoupling of the node,
- Public chain layer: This manages the public chain service. Our global public blockchain is optimized for storing hashes. Each node indexes all hashes so that they can be easily verified.

20 UI Layer

The UI layer contains two frontend applications, which enable users to easily develop and debug their Live Contracts. First is the chain viewer, which allows users to connect to a specific node and list all the chains. The user can only list and view chains of which he is a part of. The second application is the playground application, which lets the user develop Live Contract scenarios. It visualizes the scenario in a state diagram and contains other visualization and verification tools.

21 Application Layer

21.1 Web server

The web server application serves as a proxy between the frontend and the applications within the node. The web server performs two functions, listed as follows:

- Authentication of all the requests to the services
- Proxies all the request to the correct service

21.2 Workflow engine

The actual creation and execution of the Live Contracts are done by the workflow service. If an event received by the event chain service contains an action in the Live Contract it will be sent to the workflow service. The workflow service will then execute the action which will lead to a state transition in the workflow and a new projection of the workflow. This projection is stored in a MongoDB database.

22 Private chain layer

The private chain layer decouples the node. Decoupling ensures a stable system, even in case of bad connectivity or a high load. The message queue is the communication layer for the private chain. The technology providing the message queue is RabbitMQ, which is a lightweight message broker that is perfectly suited to deliver messages within the node as well as to other nodes. RabbitMQ has a function called the Shovel, which dynamically sets up a connection with another RabbitMQ broker and exchanges messages. This mechanism is used to transfer events from one node to another.

Three services manage all inbound and outbound events.

22.1 Event chain service

The service that manages the private chains is the event chain service. This service processes all incoming events. The following steps are followed while processing these events:

- Validates the incoming event(s) by checking if it's correctly signed and if the chain isn't broken.
- Validates if the chain matches the locally stored chain. If not, it performs conflict resolution wherever possible.
- If the event(s) are sent by the identity that belongs to this node, it will execute the received event(s). Otherwise, it will only store them in the database.
- If a new identity is added from a different node, the whole chain is forwarded to this node
- All new events are forwarded to the related nodes.

For storage, the event chain service uses a MongoDB database.

22.2 Event enqueue service

The event enqueue service is assigned the small task of putting events on the event queue. It does this both for services within the node (e.g. the workflow Service and the event chain service) as well as for external users.

22.3 Event dispatch service

All messages on the event queue are handled by the event dispatch Service. It picks up all the messages and distributes it to the event service. If the event service processes the message, it will be marked as handled; otherwise, it will be moved to the dead-letter queue.

23 Public chain layer

23.1 Anchor service

The anchor service is at the heart of the public chain. The anchor service will be a fork of the NXT platform extended with NG protocol. The anchor service will be extended so that it can handle both ”normal” transactions and data transactions.

These data transactions will be used to store hashes from events from the private chains, as is described in Section 16.1. All these hashes will be collected daily and merged deterministically into a Merkle tree. This way, the data transactions can be removed from storage to reduce the storage footprint, but people can still verify whether the hash existed.

To be able to verify whether a certain hash is stored, all hashes will be indexed. This makes verification much faster because there won't be any need to search through all data transactions.

24 Container orchestration

As the node comprises multiple microservices, a container orchestration platform is required to manage the running of the containers. Container orchestration platforms take care of a few tasks, such as provisioning hosts, instantiating containers, restarting failed containers and scaling the cluster by adding or removing containers. Different container orchestration tools can be used for this purpose, such as Docker Swarm, Mesos, Nomad or Kubernetes. Initially, a configuration file will be included for Kubernetes. Each service will be configured to run in its own Pod with its own load balancer. This is done so that individual services can scale independently of each other. The Horizontal Pod Autoscaler is used to manage scaling of services[92].

REFERENCES

[1] Hannah Ritchie Max Roser. *Technological Progress*. https://www.ft.com/content/cb56d86c-88d6-11e7-afd2-74b8ecd34d3b. Accessed: 03-06-2018. 2017.

[2] Christine Legner and Kristin Wende. "The challenges of inter-organizational business process design – a research agenda". In: (2007).

[3] Benjamin E. Hermalin and Michael L. Katz. "Moral Hazard and Verifiability: The Effects of Renegotiation in Agency". In: (1990).

[4] Audun Jøsang. "The right type of trust for distributed systems". In: *Proceedings of the 1996 workshop on New security paradigms*. ACM. 1996, pp. 119–131.

[5] Israel Z Ben-Shaul and Gail E Kaiser. "A paradigm for decentralized process modeling and its realization in the oz environment". In: *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press. 1994, pp. 179–188.

[6] Ray Fisman and Roberta Gatti. "Bargaining for bribes: The role of institutions". In: *International handbook on the economics of corruption* (2006), pp. 127–139.

[7] Jörg Becker, Michael Rosemann, and Christoph Von Uthmann. "Guidelines of business process modeling". In: *Business Process Management*. Springer, 2000, pp. 30–49.

[8] Manfred Reichert, Thomas Bauer, and Peter Dadam. "Enterprise-wide and cross-enterprise workflow management: Challenges and research issues for adaptive workflows". In: (1999).

[9] Vitalik Buterin. "Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform". In: (2013).

[10] Vitalik Buterin and Karthik Gollapudi. *A Next-Generation Smart Contract and Decentralized Application Platform*. https://github.com/ethereum/wiki/wiki/White-Paper/f18902f4e7fb21dc92b37e8a0963eec4b3f4793a. Accessed: 22-05-2018.

[11] Ian Grigg. "The Ricardian Contract". In: (2004).

[12] Nick Szabo. "Formalizing and Securing Relationships on Public Networks". In: (1997).

[13] Telser. "A Theory of Self-Enforcing Agreements". In: (1980).

[14] David Joulfaian Douglas Holtz-Eakin and Harvey S. Rosen. "Sticking it out: entrepreneurial survival and liquidity constraints". In: (1993).

[15] *Toshi wallet now supports ERC20 tokens and ERC721 collectibles*. https://blog.toshi.org/toshi-wallet-now-supports-erc20-tokens-and-erc721-collectibles-e718775895aa. Accessed: 30-08-2018.

[16] *ERC-20 Token Standard*. https://eips.ethereum.org/EIPS/eip-20. Accessed: 30-08-2018.

[17] Oraclize. "A Scalable Architecture for On-Demand, Untrusted Delivery of Entropy". In: (2015).

[18] Daniel IA Cohen and Daniel IA Cohen. *Introduction to computer theory*. Vol. 2. Wiley New York, 1991.

[19] Marko Vukolić. "Rethinking permissioned blockchains". In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 3–7.

[20] AbdulSalam Kalaji, Rob Mark Hierons, and Stephen Swift. "A search-based approach for automatic test generation from extended finite state machine (EFSM)". In: *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009. TAIC PART'09*. IEEE. 2009, pp. 131–132.

[21] M.G. Gouda, E.G. Manning, and Y.T. Yu. "On the Progress of Communication between Two Finite State Machines". In: (1984).

[22] G Pace and J Schapachnik. "Contracts for Interacting Two-Party Systems". In: (2012).

[23] Mark D. Flood and Oliver R Goodenough. "Contract as Automaton: The Computational Representation of Financial Agreements". In: (2015).

[24] Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari. "Automata for Service Contracts". In: (2014).

[25] Pierpaolo Degano Davide Basile and Gian-Luigi Ferrari. "From Orchestration to Choreography through Contract Automata". In: (2014).

[26] Ian Ayres and Robert Gertner. "Filling Gaps in Incomplete Contracts: An Economic Theory of Default Rules". In: (1989).

[27] Jan L.G. Dietz. "Understanding and Modeling Business Processes with DEMO". In: (1999).

[28] YoungJoon Byun, Beverly A. Sanders, and Chang-Sup Keum. "Design Patterns of Communicating Extended Finite State Machines in SDL". In: (2001).

[29] Petri. "Kommunikation mit Automaten". In: (1962).

[30] Dennis Kafura. *Notes on Petri Nets*. http://people.cs.vt.edu/kafura/ComputationalThinking/Class-Notes/Petri-Net-Notes-Expanded.pdf. Accessed: 01-09-2018.

[31] Wil M.P. van der Aalst. "The Application of Petri Nets to Workflow Management". In: (1998).

[32] Jan Recker et al. "How good is bpmn really? Insights from theory and practice". In: (2006).

[33] Wil M.P. van der Aalst et al. "Life After BPEL?" In: (2005).

[34] Luciano García-Bañuelos et al. "Optimized Execution of Business Processes on Blockchain". In: (2017).

[35] Jan L.G. Dietz. "DEMO: Towards a discipline of organisation engineering". In: (1999).

[36] *JSONForms - React*. https://jsonforms.io/. Accessed: 30-08-2018.

[37] *JSONForm - Bootstrap 3*. https://github.com/jsonform/jsonform. Accessed: 30-08-2018.

[38] *Mozilla react-jsonschema-form*. https://github.com/mozilla-services/react-jsonschema-form. Accessed: 30-08-2018.

[39] *Angular Schema Form*. http://schemaform.io/. Accessed: 30-08-2018.

[40] *Open Document Format*. http://www.opendocumentformat.org/. Accessed: 30-08-2018.

[41] Sindhu Sajana and Sethumadhavan. "On Blockchain Applications: Hyperledger Fabric And Ethereum". In: (2018).

[42] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158.

[43] Daniel Brand and Pitro Zafiropulo. "On communicating finite-state machines". In: *Journal of the ACM (JACM)* 30.2 (1983), pp. 323–342.

[44] Robert M Hierons AbdulSalam Kalaji and Stephen Swift. "New approaches for passive testing using an Extended Finite State Machine specification". In: (2003).

[45] O Sury and R Edmonds. *Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC*. Tech. rep. 2017.

[46] NIST. *Transition Plans for Key Establishment Schemes using Public Key Cryptography*. https://csrc.nist.gov/News/2017/Transition-Plans-for-Key-Establishment-Schemes. Accessed: 13-07-2018. 2017.

[47] Daniel J. Bernstein et al. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (2012),

pp. 77–89. ISSN: 2190-8516. DOI: 10.1007/s13389-012-0027-1. URL: https://doi.org/10.1007/s13389-012-0027-1.

[48] Henri Gilbert and Helena Handschuh. "Security analysis of SHA-256 and sisters". In: *International workshop on selected areas in cryptography*. Springer. 2003, pp. 175–193.

[49] NIST. *NIST Policy on Hash Functions*. https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions. Accessed: 13-07-2018. 2015.

[50] Bruce Schneier and John Kelsey. "Cryptographic Support for Secure Logs on Untrusted Machines." In: *USENIX Security Symposium*. Vol. 98. 1998, pp. 53–62.

[51] E. Brewer. "Towards Robust Distributed System. Symposium on Principles of Distributed". In: (2000).

[52] Peter Bailis and Ali Ghodsi. "Eventual consistency today: Limitations, extensions, and beyond". In: *Queue* 11.3 (2013), p. 20.

[53] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[54] Miguel Castro and Barbara Liskov. *Byzantine fault tolerance*. US Patent 6,671,821. 2003.

[55] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. Accessed: 17-05-2018.

[56] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[57] POA Network. *Proof of Authority: consensus model with Identity at Stake.* https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256. Accessed: 17-05-2018.

[58] Marko Vukolic. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: (2016).

[59] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[60] Git Documentation. *Git Branching - Rebasing*. https://git-scm.com/book/en/v2/Git-Branching-Rebasing. Accessed: 09-08-2018.

[61] Aaron van Wirdum. "Rejecting Today's Hard Fork, the Ethereum Classic Project Continues on the Original Chain: Here's Why". In: *Bitcoin Magazine* 20 (2016).

[62] European Parliament. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL: on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).* https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN. Accessed: 12-07-2018. 2016.

[63] Olly Jackson. "Is it possible to comply with GDPR using blockchain?" In: *International Financial Law Review* (2018).

[64] *Art. 28 GDPR - Processor*. https://gdpr-info.eu/art-28-gdpr/. Accessed: 15-09-2018.

[65] S Goldwasser, S Micali, and C Rackoff. "The knowledge complexity of interactive proof systems". In: (1989).

[66] *Blockchain costs per transaction*. https://www.blockchain.com/charts/cost-per-transaction. Accessed: 05-09-2018.

[67] Emanuel Palm. *Implications and Impact of Blockchain Transaction Pruning*. 2017.

[68] Wenting Li et al. "Towards scalable and private industrial blockchains". In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 9–14.

[69] Serguei Popov. "A probabilistic analysis of the nxt forging algorithm". In: *Ledger* 1 (2016), pp. 69–83.

[70] Waves platform. *WAVES whitepaper*. https://blog.wavesplatform.com/waves-whitepaper-164dd6ca6a23. Accessed: 16-07-2018. 2016.

[71] *Chainpoint Node API: How to Create a Chainpoint Proof*. https://github.com/chainpoint/chainpoint-node/wiki/Chainpoint-Node-API:-How-to-Create-a-Chainpoint-Proof. Accessed: 05-09-2018.

[72] Gleb Kostarev. *Review of blockchain consensus mechanisms*. https://blog.wavesplatform.com/review-of-blockchain-consensus-mechanisms-f575afae38f2. Accessed: 13-07-2018. 2017.

[73] NEM. *NEM technical reference*. https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf. Accessed: 13-07-2018. 2018.

[74] LTO. "LTO Token Economy". In: (2018).

[75] Waves Platform. *Blockchain Leasing For Proof Of Stake*. https://blog.wavesplatform.com/blockchain-leasing-for-proof-of-stake-bac5335de049. Accessed: 13-07-2018. 2018.

[76] mthcl. "The math of Nxt forging". In: (2014).

[77] *Waves generators*. http://dev.pywaves.org/generators/. Accessed: 28-08-2018.

[78] *Nxt Blockchain Explorer*. https://nxtportal.org/monitor/. Accessed: 28-08-2018.

[79] Kofman Begicheva. "Fair Proof of Stake". In: (2018).

[80] *Waves-NG stress test: results in!* https://blog.wavesplatform.com/waves-ng-stress-test-results-in-44090f59bb15. Accessed: 05-09-2018.

[81] S. Haber and W.S. J Stornetta. "How to time-stamp a digital document". In: (1991).

[82] Ralph C. Merkle. "Method of providing digital signatures". U.S. pat. 4309569. Jan. 5, 1982.

[83] A. Begicheva and I. Smagin. "RIDE: a Smart Contract Language for Waves". Pat. 2018.

[84] Saifedean Ammous. "Blockchain Technology: What is it good for?" In: (2016).

[85] *Blockchain number of transaction*. https://www.blockchain.com/en/charts/n-transactions-total. Accessed: 05-09-2018.

[86] *BIP 141: Segregated Witness (Consensus layer)*. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. Accessed: 05-09-2018.

[87] *Problems Ethereum*. https://github.com/ethereum/wiki/wiki/Problems. Accessed: 30-08-2018.

[88] G Hardin. "The Tragedy of the Common". In: (1969).

[89] *Nothing considered a look at nothing at stake vulnerability for cryptocurrencies*. https://pivx.org/nothing-considered-a-look-at-nothing-at-stake-vulnerability-for-cryptocurrencies/. Accessed: 30-08-2018.

[90] John R Douceur. "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[91] Marc Stevens et al. "The first collision for full SHA-1". In: (2017).

[92] Kubernetes. *Kubernetes, Horizontal Pod Autoscaling*. https://github.com/kubernetes/community/blob/master/contributors/design-proposals/autoscaling/horizontal-pod-autoscaler.md. Accessed: 03-08-2018.

#	Field Name	Length
1	Version	1
2	Timestamp	8
3	Parent block signature	64
4	Consensus block length	4
5	Base target	8
6	Generation Signature	32
7	Transaction list Hash	32
8	Anchor Merkle root	32
9	Generator public key	32
10	Block's signature	64

TABLE 1: Key block structure

#	Field Name	Length
1	Amount of transactions (x)	4
2	Transaction #1 bytes	113
...	...	...
2 + (K - 1)	Transaction #K bytes	113

TABLE 2: Key block attachment

#	Field Name	Length
1	Version	1
2	Timestamp	8
3	Parent block signature	64
4	Consensus block length	4
5	Base target	8
6	Generation Signature	32
7	Transaction list Hash	32
8	Transaction #1 bytes	TODO
...	...	...
$8 + (K - 1)$	Transaction #K bytes	TODO
$9 + (K - 1)$	Balance change summary entry #1	40 (Table 4)
...	...	...
$9 + (K - 1) + (N - 1)$	Balance change summary entry #N	40 (Table 4)
$10 + (K - 1) + (N - 1)$	Generator public key	32
$11 + (K - 1) + (N - 1)$	Block's signature	64

TABLE 3: Summary Block structure

#	Field Name	Length
2	Wallet address	32
3	Balance change	8

TABLE 4: Balance summary entry

#	Field Name	Length
1	Transaction type	1
2	Anchor hash	32
3	Fee	8
4	Timestamp	8
5	Signature	64

TABLE 5: Anchor transactions structure

#	Field Name	Length
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

TABLE 6: Transfer transaction structure

#	Field Name	Length
1	Transaction type	1
2	Id	32
3	Sending address	32
4	Receiving address	32
5	Expiration Date	8
6	Certificate Type	32
7	Fee	8
8	Timestamp	8
9	Signature	64

TABLE 7: Issue certificate transaction structure

#	Field Name	Length
1	Transaction type	1
2	Id	32
3	New expiration Date	8
4	Fee	8
5	Timestamp	8
6	Signature	64

TABLE 8: Update certificate transaction structure

#	Field Name	Length
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

TABLE 9: Lease transaction structure

#	Field Name	Length
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

TABLE 10: Cancel Lease transaction structure