

Maurizio Pizzonia Andrea Vitaletti (Eds.)

DLT 2022
4th Workshop on
Distributed Ledger Technology

June 20, 2022

Rome, Italy

Workshop co-located with ITASEC 2022

Proceedings

Copyright © 2022 for the individual papers by the papers' authors.
Copyright © 2022 for the volume as a collection by its editors.

This volume and its papers are published under license “Creative Commons Attribution 4.0 International” (CC BY 4.0).



Editors' addresses:

Maurizio Pizzonia
Università degli Studi Roma Tre
Dipartimento di Ingegneria
Via della Vasca Navale 79
00146, Rome, Italy
maurizio.pizzonia@uniroma3.it

Andrea Vitaletti
Sapienza Università di Roma
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Via Ariosto 25
00185, Rome Italy
vitaletti@diag.uniroma1.it

Preface

This book contains the contributions which were selected for publication at the fourth edition of the Distributed Ledger Technology Workshop (DLT 2022), which has been held in conjunction with ITASEC22 in Rome, Italy, on June 20, 2022. This event follows the first three editions of the workshop held at Perugia in 2018, at Pisa in 2019 and at Ancona in 2020, respectively, and represents the annual meeting of the Italian DLT group.

The last years have witnessed an impressive and increasing interest around Distributed Ledger Technology. A huge number of application fields, including finance, academics, IoT, industries, just to mention some of the most popular ones, are experiencing the advantages of reliable and unalterable information storage and exchange without any trusted third party. This large interest makes discussion on open problems more pressing, like, for example, regarding privacy, scalable and low-latency architectures, interoperability, off-chain solutions, legal aspects, etc. Intersections of blockchains with other technological trends, like, AI, big data, IoT, Industry 4.0, are further interesting discussion topics.

The primary goal of DLT2022 was to foster discussion and cross-fertilisation of ideas among experts in different fields related to DLTs, and thus advance the international state-of-the-art. Similarly to the previous editions, the DLT 2022 workshop solicited two kinds of contributions: research papers and oral communications. Both types of contribution entailed an oral presentation at the workshop, but only the former ones are reported in this book. In particular, the workshop accepted 10 research papers and 6 oral contributions.

We would like to express our thanks to the authors who submitted their papers to the workshop, and to the members of the Technical Program Committee for their valuable work in evaluating the submitted papers.

June 2022

Maurizio Pizzonia
Andrea Vitaletti

Organizing Committee

Diego Pennino, Università degli Studi Roma Tre
Marco Zecchini, Università degli Studi di Roma "La Sapienza"

Program Committee

Leonardo Aniello, University of Southampton
Marco Baldi, Università Politecnica delle Marche
Andrea Bracciali, University of Stirling
Francesco Buccafurri, Università degli Studi "Mediterranea" di Reggio Calabria
Jing Chen, Stony Brook University
Franco Chiaraluca, Università Politecnica delle Marche
Stelvio Cimato, Università degli Studi di Milano
Gabriele D'Angelo, Università di Bologna
Andrea De Salve, Consiglio Nazionale delle Ricerche, Italy
Damiano Di Francesco Maesa, Consiglio Nazionale delle Ricerche, Italy
Changyu Dong, Newcastle University
Stefano Ferretti, Università di Bologna
Letterio Galletta, IMT Scuola Alti Studi Lucca
Alberto Leporati, Università degli Studi di Milano
Michele Marchesi, Università degli Studi di Cagliari
Giorgia Azzurra Marson, NEC Laboratories Europe GmbH
Paolo Mori, Istituto di Informatica e Telematica del CNR
Andrea Morichetta, Università degli Studi di Camerino
Leonardo Mostarda, Università degli Studi di Camerino
Jose Luis Muñoz Tapia, Universitat Politècnica de Catalunya
Guillermo Navarro-Arribas, Universitat Autònoma de Barcelona
Remo Pareschi, Università degli Studi del Molise
Laura Ricci, Università di Pisa
Giovanni Sartor, EUI/CIRSFID
Ivan Visconti, Università di Salerno
Roberto Zunino, Università degli Studi di Trento

Contents

Analysis of a Blockchain Protocol Based on LDPC Codes <i>Massimo Battaglioni, Paolo Santini, Giulia Rifaiani, Franco Chiaraluce and Marco Baldi</i>	7
Evaluating Blockchain Systems: A Comprehensive Study of Security and Dependability Attributes <i>Stefano De Angelis, Gilberto Zanfino, Leonardo Aniello, Federico Lombardi and Vladimiro Sassone</i>	18
A Study on Diem Distributed Ledger Technology <i>Giuseppe Antonio Pierro and Roberto Tonelli</i>	33
DELTA Distributed Elastic Log Text Analyser <i>Pierluigi Di Pilla, Remo Pareschi, Francesco Salzano and Federico Zappone</i>	48
A Provably-Unforgeable Threshold Schnorr Signature With an Offline Recovery Party <i>Michele Battagliola, Alessio Galli, Riccardo Longo and Alessio Meneghetti</i>	60
Cob: A Consensus Layer Enabling Sustainable Sharding-Based Consensus Protocols <i>Andrea Flamini, Riccardo Longo and Alessio Meneghetti</i>	77
Olive Oil as Case Study for the *-Chain Platform <i>Stefano Bistarelli, Francesco Faloci, Paolo Mori and Carlo Taticchi</i>	94
AgriChain: Blockchain Syntactic and Semantic Validation for Reducing Information Asymmetry in Agri-Food <i>Pierluigi Gallo, Federico Daidone, Filippo Sgroi and Mirko Avantageggiato</i>	103
On-Chain Global Maintenance Services <i>Alessandro Bellini, Antonio Bonifacio, Salvatore Esposito De Falco, Simone Naldini, Francesco Pacileo, Diego Pennino, Maurizio Pizzonia, Domenico Sardanelli, Andrea Vitaletti, Pietro Vito and Marco Zecchini</i>	119
Blockchain-Based Tracking of the Supply Chain of the Italian Craft Beer Sector <i>Lorenzo Ariemma, Niccolò De Carlo, Diego Pennino, Maurizio Pizzonia, Andrea Vitaletti and Marco Zecchini</i>	130

Analysis of a Blockchain Protocol Based on LDPC Codes

Massimo Battaglioni^{1,*}, Paolo Santini¹, Giulia Rafaiani¹, Franco Chiaraluce¹ and Marco Baldi¹

¹Department of Information Engineering, Università Politecnica delle Marche, Ancona, 60131, Italy

Abstract

In a blockchain Data Availability Attack (DAA), a malicious node publishes a block header but withholds part of the block, which contains invalid transactions. Honest full nodes, which can download and store the full ledger, are aware that some data are not available but they have no formal way to prove it to light nodes, i.e., nodes that have limited resources and are not able to access the whole blockchain data. A common solution to counter these attacks exploits linear error correcting codes to encode the block content. A recent protocol, called SPAR, employs coded Merkle trees and low-density parity-check codes to counter DAAs. In this paper, we show that the protocol is less secure than claimed, owing to a redefinition of the adversarial success probability. As a consequence we show that, for some realistic choices of the parameters, the total amount of data downloaded by light nodes is larger than that obtainable with competing solutions.

Keywords

Blockchain, data availability attacks, LDPC codes, SPAR protocol

1. Introduction

A blockchain can be seen as an ordered list of blocks, each containing a set of transactions occurred among the participants of a peer-to-peer network. The recent discovery of Data Availability Attacks (DAAs) represents a new threat against blockchain security. Since the DAA introduction in [2], there has been a growing research interest in finding efficient countermeasures to this type of attacks, possibly leading to new blockchain models with improved scalability and security (e.g., [13, 8, 1, 5]).

In fact, scalability, which is related to the ability of supporting large transaction rates, represents one of the main issues in most existing blockchains [14]. The straightforward solution of increasing the block size raises a series of further concerns. In fact, the larger the block size the smaller the number of nodes able to download the full blockchain and, indeed, to participate in the network as *full nodes*, verifying the validity of new blocks and of every contained transaction. More peers would rather participate in the network as *light nodes*, which, due to

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

✉ m.battaglioni@univpm.it (M. Battaglioni); p.santini@univpm.it (P. Santini); g.rafaiani@univpm.it (G. Rafaiani); f.chiaraluce@univpm.it (F. Chiaraluce); m.baldi@univpm.it (M. Baldi)

ORCID 0000-0002-8539-4007 (M. Battaglioni); 0000-0003-0631-3668 (P. Santini); 0000-0003-0029-5104 (G. Rafaiani); 0000-0001-6994-1448 (F. Chiaraluce); 0000-0002-8754-5526 (M. Baldi)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

their limited resources, store only a squeezed version of the blockchain [10] and, consequently, cannot autonomously verify the validity of transactions.

Light nodes aim at downloading as less data as possible. For instance, they may store only the block headers, which unambiguously identify the content of the blocks. However, in a setting with relatively few full nodes, collusion among them is more probable; this makes light nodes more susceptible to DAAs. In fact, the aim of a DAA is to make at least one light node accept a block which has not been fully disclosed to the network. This can happen if and only if honest full nodes are prevented from preparing *fraud proofs*, *i.e.*, demonstrations that the block is invalid [13, 3].

One of the most promising countermeasures to DAAs consists in encoding the blocks through some error correcting code. Encoding introduces redundancy and distributes the information of each transaction across all the codeword symbols, so that recovering a small portion of an encoded block may be enough to retrieve the entirety of its contents through decoding. This strategy, combined with a sampling process in which light nodes ask for fragments of an encoded block and then gossip them to the connected full nodes, ensures that malicious block producers are forced to reveal enough pieces of the invalid block [3]. An alternative to transactions encoding is to change the protocol in such a way that a group of light nodes can collaboratively (among themselves) and autonomously (from full nodes) verify blocks [5]. Another option is to decouple the consensus rules from the transaction validity rules [1, 4].

In a recent paper [13], Yu et al. proposed SPAR, a blockchain protocol which uses Low-Density Parity-Check (LDPC) codes to counter DAAs; LDPC codes for this specific application have then been studied in [8, 9]. SPAR comes as an improvement of the protocol in [3], the latter using two-dimensional Reed-Solomon codes, whose parameters have been optimized in [12]. The authors of SPAR study the protection against DAAs in case the adversary aims to prevent honest full nodes from successfully decoding the block, which is a strict requirement to settle a proper fraud proof. In [13], this situation is investigated assuming the adversary operates by withholding pieces of the encoded block; under a coding theory perspective, this gets modeled as a transmission over an erasure channel. They conclude that, unless the adversary is able to find stopping sets (which is an NP-hard problem [6]), SPAR guarantees that the success probability of a DAA is sufficiently small even when light nodes download a small amount of data besides the block header. As a consequence, SPAR claims improvements in all the relevant metrics [13, Table 1].

Our contribution In this paper we study the security of the SPAR protocol. Namely, we recompute the adversarial success probability with the consideration that deceiving *at least* a single light node is a success for the attacker, which is the same scenario considered in [3]. This yields a sampling cost that is much larger than the expected one, thus penalizing the light nodes participating in the network. Moreover, we show that the total amount of data that light nodes have to download (header size plus sampling cost) is actually larger than that of competing solutions such as [3].

Paper organization The paper is organized as follows. In Section 2 we describe the notation and some background. In Section 3 we introduce a general framework to study DAAs. In Section

4 we provide some numerical results. Finally, in Section 5 we draw some conclusions.

2. Notation and background

In this section we establish the notation used throughout the paper, and recall some background notions.

2.1. Mathematical notation

Given two integers a and b , we use $[a, b]$ to indicate the set of integers x such that $a \leq x \leq b$. For a set A , we use $|A|$ to denote its cardinality. We denote with \mathbb{F}_q the finite field with q elements. Given a vector \mathbf{v} , we use $\text{supp}(\mathbf{v})$ to denote its support, *i.e.*, the set containing the positions of its non-zero entries and $w_H(\mathbf{v})$ to denote its Hamming weight, that is, the size of its support. Given an integer l and a set A , A^l is the set of vectors of length l taking entries in A . Given a matrix \mathbf{M} , $m_{i,j}$ denotes its entry at row i and column j , $\mathbf{M}_{i,:}$ denotes the i -th row, and $\mathbf{M}_{:,j}$ denotes the j -th column. Given a set A , $\mathbf{M}_{:,A}$ (respectively, $\mathbf{M}_{A,:}$) represents the matrix formed by the columns (respectively, rows) of \mathbf{M} indexed by A .

We denote by Concat the string concatenation function and by $H_b(\cdot)$ the binary entropy function. Moreover, we denote by Hash a cryptographic hash function, with codomain D . Given some vector \mathbf{a} , we use $\mathcal{T}(\mathbf{a})$ to denote a generic hash tree structure constructed from \mathbf{a} and using Hash as underlying function. The root of the tree is denoted as $\mathcal{T}.\text{Root}(\mathbf{a})$; it generically takes values in D^t and is a one-way function. With analogous notation, by $\mathcal{T}.\text{Proof}(\mathbf{a}, i)$ we refer to the proof that the i -th entry of \mathbf{a} is a leaf in the base layer of the tree. Notice that, when Hash is properly chosen, then for any pair of strings $\mathbf{a} \neq \mathbf{a}'$ we have $\mathcal{T}.\text{Root}(\mathbf{a}) \neq \mathcal{T}.\text{Root}(\mathbf{a}')$ and, for any index i , $\mathcal{T}.\text{Proof}(\mathbf{a}, i) \neq \mathcal{T}.\text{Proof}(\mathbf{a}', i)$ with overwhelming probability (say, not lower than $1 - 2^{-256}$ for modern hash functions); therefore, for the sake of simplicity, in the following we assume the absence of root and proof collisions.

2.2. LDPC codes

LDPC codes are a family of linear codes characterized by parity-check matrices having a relatively small number of non-zero entries compared to the number of zeros. Namely, if an LDPC $\mathbf{H} \in \mathbb{F}_q^{r \times n}$ has full rank $r < n$ and row and column weight in the order of $\log(n)$ and $\log(r)$, respectively, then it defines an LDPC code with length n and dimension $k = n - r$, with code rate $R = \frac{k}{n}$. If all the rows (columns, respectively) of \mathbf{H} have the same Hamming weight, we denote it as w (v , respectively). The associated code is $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}\mathbf{H}^\top = \mathbf{0}\}$, where $^\top$ denotes transposition. The rows of the parity-check matrix define the code *parity-check equations*, that is,

$$\sum_{j=1}^n c_j h_{i,j} = 0, \quad \forall i \in [1, r], \quad \forall \mathbf{c} \in \mathcal{C}. \quad (1)$$

Equivalently, any code can be represented in terms of a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, which forms a basis for \mathcal{C} .

In an Erasure Channel (EC), some of the codeword symbols are replaced with the erasure symbol ϵ . To this end, we express the action of an EC as $\mathbf{c} + \mathbf{e}'$, where \mathbf{c} is the input sequence and $\mathbf{e}' \in \{0, \epsilon\}^n$, with ϵ such that, conventionally, $\epsilon + a = \epsilon, \forall a \in \mathbb{F}_q$. A decoding algorithm for the EC aims to obtain a codeword by substituting each erasure with an element from \mathbb{F}_q . In the case of LDPC codes, the most common decoder used over the EC is the peeling decoder [7]. This algorithm works by expressing (1) as a linear system, where the unknowns are exactly the erased symbols. Due to the sparsity of \mathbf{H} , with large probability the linear system will include several univariate equations, *i.e.*, containing only one erasure. Each of these equations can be solved to compute the corresponding unknown, which is then substituted into all the other equations. This procedure is iterated until all the unknowns are found or, at some point, the linear system does not contain any univariate equation, *i.e.*, all the unsolved equations contain at least two unknowns. In the former case we have a decoding success, while in the latter case we have a failure, due to a *stopping set* [11], *i.e.*, a set of symbols participating to parity-check equations that contain at least two unknowns each. If all the symbols forming a stopping set are erased, peeling decoding fails. The *stopping ratio* β of an LDPC code is defined as the minimum stopping set size divided by n .

2.3. Components of the SPAR protocol

SPAR is based on a novel hash tree called Coded Merkle Tree (CMT), combined with an ad-hoc *hash-aware* peeling decoder.

Coded Merkle Tree A CMT is a hash tree which is constructed from ℓ linear codes $\{\mathcal{C}^{(1)}, \dots, \mathcal{C}^{(\ell)}\}$ over \mathbb{F}_q ; the i -th code has length n_i and dimension k_i . Each code $\mathcal{C}^{(i)}$ is defined by the systematic generator matrix $\mathbf{G}^{(i)} = [\mathbf{I}_{k_i} \mid \mathbf{A}_i]$, with $\mathbf{A}_i \in \mathbb{F}_q^{k_i \times (n_i - k_i)}$ and \mathbf{I}_{k_i} being the identity matrix of size k_i . The CMT uses an integer b which must be a divisor of all codeword length values n_1, \dots, n_ℓ . Furthermore, one needs to have partitions for the sets $[1, n_i]$, for $i \in [1, \ell - 1]$. Namely, we have $\mathcal{S}_i = \{S_1^{(i)}, \dots, S_{k_{i+1}}^{(i)}\}$ which is a partition of $[1, n_i]$, such that the $S_j^{(i)}$ are all disjoint and each one contains b elements, since $k_{i+1} = n_i/b$. Starting from $\mathbf{c} \in \mathcal{C}^{(1)}$, we build the associated CMT $\mathcal{T}'(\mathbf{c})$ as follows:

1. set $i = 1$;
2. for $j \in \{1, \dots, k_{i+1}\}$, set

$$u_j = \text{Concat} \left(\text{Hash} \left(\mathbf{c}_{z_1}^{(i)} \right), \dots, \text{Hash} \left(\mathbf{c}_{z_b}^{(i)} \right) \right),$$

with $\{z_1, \dots, z_b\} = S_j^{(i)}$;

3. encode $\mathbf{u} = [u_1, \dots, u_{k_{i+1}}]$ as¹ $\mathbf{c} = \mathbf{u}\mathbf{G}^{(i+1)}$;
4. if $i < \ell - 1$, increase i and restart from step 2), otherwise set $\mathcal{T}'.\text{Root}(\mathbf{c}) = \mathbf{u}$.

¹Notice that, when LDPC codes are considered, encoding is conveniently performed using the parity-check matrix rather than the generator matrix. This implementation detail does not affect the conclusions of our analysis but, considering encoding with the parity-check matrix, we would unnecessarily burden the notation. Therefore, we stick to encoding with the generator matrix.

Hash-aware peeling decoder A hash-aware peeling decoder, described in [13, Section 4.3], is an algorithm that decodes a set of ℓ words which are expected to constitute a CMT. Namely, let $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\ell)}\}$, where $\mathbf{x}^{(i)} \in \{\mathbb{F}_q \cup \{\epsilon\}\}^{n_i}$, be the words to be decoded. The hash-aware peeling decoder works in a top-down fashion and, at every iteration, uses the peeling decoder strategy (*i.e.*, recover erasures that participate in univariate parity-check equations) for any layer of the CMT. Additionally, the hash-aware peeling decoder verifies the consistency between symbols of connected layers of the tree via hash functions, whilst the symbols are recovered. Decoding fails whenever a stopping set or a failed parity-check equation is met, just like the conventional peeling decoder. Furthermore, the hash-aware peeling decoder fails in case check consistency fails for some layer. Finally, an undetected error is met (but not recognized by the decoder) if the decoded sequence is a codeword, but not the original one.

3. A general framework to study DAAs

In this section we present a general framework to study DAAs, and then apply it to the SPAR protocol. For brevity, we only give the fundamentals of the model; for further details concerning DAAs, we refer the interested reader to [13, 3].

3.1. A general model for DAAs

We consider a game in which an adversary \mathcal{A} exchanges messages with m players $\mathcal{P}_1, \dots, \mathcal{P}_m$, who cannot communicate among themselves. Each player has access to an oracle \mathcal{O} , who can only perform polynomial time operations. Every list of transactions is seen as a vector $\mathbf{u} \in \mathbb{F}_q^k$. We assume that the following information is publicly available:

- a validity function $f : \mathbb{F}_q^k \mapsto \{\text{False}, \text{True}\}$, which depends on the blockchain rules and on its current status;
- a k -dimensional code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with generator matrix \mathbf{G} ;
- two hash trees $\mathcal{T}, \mathcal{T}'$ (the former is constructed upon the uncoded data \mathbf{u} , while the latter is constructed upon the codeword $\mathbf{c} = \mathbf{u}\mathbf{G}$).

The game proceeds as follows:

1. \mathcal{A} chooses $\mathbf{u} \in \mathbb{F}_q^k$ such that $f(\mathbf{u}) = \text{False}$ and a vector $\tilde{\mathbf{c}} \in \mathbb{F}_q^n$;
2. \mathcal{A} challenges the players with (h_u, h_c) , where $h_u = \mathcal{T}.\text{Root}(\mathbf{u})$, $h_c = \mathcal{T}'.\text{Root}(\tilde{\mathbf{c}})$;
3. each player \mathcal{P}_i selects $J_i \subseteq [1, n]$ with size s ;
4. \mathcal{A} receives $U = \bigcup_{i=1}^m J_i$;
5. to reply to a query containing the index i , \mathcal{A} must send $\{\tilde{c}_i, \mathcal{T}.\text{Proof}(\tilde{\mathbf{c}}, i)\}$; \mathcal{A} is free to choose which queries to reply and which ones to ignore;
6. if a player does not receive a valid reply for any of his queries, then he discards (h_u, h_c) ;
7. the players gossip all the valid answers to \mathcal{O} , that aims to produce a proof for one of the following facts:
 - a) $\exists \tilde{\mathbf{c}} \notin \mathcal{C}$, such that $\mathcal{T}'.\text{Root}(\tilde{\mathbf{c}}) = h_c$;
 - b) $\exists \tilde{\mathbf{c}} \in \mathcal{C}$ such that $\mathcal{T}'.\text{Root}(\tilde{\mathbf{c}}) = h_c$, $\tilde{\mathbf{c}} = \tilde{\mathbf{u}}\mathbf{G}$ and $\mathcal{T}.\text{Root}(\tilde{\mathbf{u}}) \neq h_u$;
 - c) $\exists \tilde{\mathbf{c}} \in \mathcal{C}$ such that $\mathcal{T}'.\text{Root}(\tilde{\mathbf{c}}) = h_c$, $\tilde{\mathbf{c}} = \tilde{\mathbf{u}}\mathbf{G}$, $\mathcal{T}.\text{Root}(\tilde{\mathbf{u}}) = h_u$ and $f(\tilde{\mathbf{u}}) = \text{False}$.

Let us also define two properties.

Property 1. Soundness: *if a player accepts (h_u, h_c) , then \mathcal{O} will be able to recover $\tilde{\mathbf{c}}$ (and $\tilde{\mathbf{u}}$) within a finite maximum delay.*

Property 2. Agreement: *if a player accepts (h_u, h_c) , then all the other players will accept (h_u, h_c) within a finite maximum delay.*

Clearly, if \mathcal{A} wins the game, which happens with probability γ , soundness and agreement are caused to fail. We denote by γ the Adversarial Success Probability (ASP), *i.e.*, the probability that \mathcal{A} wins a random execution of the game.

It can be easily seen that, in our model, the players $\mathcal{P}_1, \dots, \mathcal{P}_m$ correspond to the light nodes connected to a malicious node modeled by \mathcal{A} . The oracle \mathcal{O} instead represents the fact that we assume any light node must be connected to at least one honest full node wishing to broadcast fraud proofs. We remark that the hypotheses and properties that underlie our model are the same under which DAAs have been studied in the literature [13, 3, 8, 12]. Finally, our model does not fix any hash tree, nor code family; thus, it can be used to study several blockchain networks. We now proceed by describing how SPAR adapts to such a model, but it can be easily seen that also the protocol proposed in [3] fits into the model.

3.2. DAAs in the SPAR protocol

In SPAR, the CMT is instantiated using the code design procedure considered in [7], which produces an ensemble of LDPC codes whose parity-check matrices have at most column weight v and at most row weight w . As mentioned in Section 2.3, besides the CMT, SPAR requires the use of another hash tree, denoted by \mathcal{T} and considered as a standard Merkle tree.

Let $\mathbf{u} \in \mathbb{F}_q^k$ denote the list of transactions of a new block. Then, a correctly constructed header contains $h_u = \mathcal{T}.\text{Root}(\mathbf{u})$ and $h_c = \mathcal{T}.\text{Root}(\mathbf{c})$, with $\mathbf{c} = \mathbf{u}\mathbf{G}^{(1)}$. However, in case of a DAA, the word $\tilde{\mathbf{c}} = \mathbf{c} + \mathbf{e}$ upon which h_c is constructed may be any vector picked from \mathbb{F}_q^n . The authors of SPAR study the protection of the protocol against DAAs; namely, they initially consider the following two cases:

- a) if $\tilde{\mathbf{c}}^{(i)} \notin \mathcal{C}^{(i)}$, then the proof consists in sending the value of all the symbols that participate in a failed parity-check equation, except for one of them, together with their CMT proofs; we refer to such a proof as *parity-check equation incorrect-coding proof*;
- b) if $\tilde{\mathbf{c}} = \mathbf{c}$ but $f(\mathbf{u}) = \text{False}$, the adversary succeeds only if the samples received by the oracle are not enough to allow the recovery of \mathbf{u} from $\tilde{\mathbf{c}}$ through decoding.

The scenario where the oracle finds a hash inconsistency is also considered, in which case \mathcal{O} can broadcast a fraud proof to the light nodes, called here *hash inconsistency incorrect-coding proof*.

The following bound for the ASP is derived [13, Theorem 1]:

$$\gamma \leq \max \left\{ (1 - \alpha_{\min})^s, 2^{\max_i \{H_b(\alpha_i)n_i + ms \log_2(1 - \alpha_i)\}} \right\} \quad (2)$$

where α_i is the *undecodable ratio* of $\mathcal{C}^{(i)}$, that is, the minimum fraction of coded symbols the adversary needs to make unavailable in order to prevent the oracle from full decoding,

$\alpha_{\min} = \min_i(\alpha_i)$, and s is the number of queries performed by each light node. Therefore, if the oracle is not able to decode due to the presence of a stopping set, the adversarial success probability computed in [13] is the probability that *exactly* one player receives an answer to all its queries.

We argue here, instead, that a sufficient condition to break the soundness and agreement as defined in [3, 13], and recalled in Section 3.1 is actually that *at least* one player accepts an invalid block.

Proposition 1. *In SPAR, an adversary cannot cause the soundness and agreement to fail with probability*

$$\gamma \leq \min\{1, \max\{1 - [1 - (1 - \alpha_{\min})^s]^m, t_2\}\}, \quad (3)$$

where $t_2 = 2^{\max_i\{b(\alpha_i)n_i + ms \log(1 - \alpha_i)\}}$.

Proof. According to Property 1, the soundness fails if at least one player accepts the block header, but the oracle will not be able to dispatch a fraud proof. The probability that exactly one player accepts the challenge is lower than or equal to $(1 - \alpha_{\min})^s$ and, therefore, the probability that exactly one player discards the challenge is larger than or equal to $1 - (1 - \alpha_{\min})^s$. Considering that there are m players, the probability that all of them discard the block is larger than or equal to $[1 - (1 - \alpha_{\min})^s]^m$. So, finally, the probability that at least a player accepts the block is lower than or equal to

$$1 - [1 - (1 - \alpha_{\min})^s]^m.$$

The rest of the proof is as in [13, Theorem 1]. □

4. Numerical examples

Let us consider the code parameters proposed in [13] as a benchmark. It is shown in [13, Table 2] that the most favourable value of the stopping ratio of the constructed ensemble (β^*) is obtained when $w = 8$ and the code rate is $R = 1/4$, from which $v = 6$ easily follows. As in [13] we consider two cases: a *strong adversary* (SA) able to find stopping sets and erase the corresponding symbols, and a *weak adversary* (WA) unable to find them and hence forced to erase random symbols. For the SA, the undecodable ratio is $\alpha^* = \beta^* = 12.4\%$; instead, in case of WA we have $\alpha^* = 47\%$ [13]. According to [13, Table 2], when $n = 4096$, the probability that the code stopping ratio α is smaller than the ensemble stopping ratio is relatively small ($3.2 \cdot 10^{-4}$).

In Table 1 we report the upper bound (2) and the newly assessed upper bound (3) on the ASP, for some values of s , considering $n = 4096$ and $m = 1024$; notice that the new value is never smaller than the previously computed upper bound. Clearly, this may have severe security consequences.

As obvious and expected, the upper bound on ASP is a decreasing function of the number of samples s . So, once a target adversarial success probability is chosen, it is possible to compute a lower bound for the value of s each player needs to ask in order to stay below it, simply by inverting (2) and (3). Considering the same parameters as above ($n = 4096$ and $m = 1024$) we obtain the results in Table 2.

Table 1Upper bound values from (2) and (3) for $m = 1024$, $n = 4096$.

s	Upper bound on γ [13]		New upper bound on γ	
	WA (2)	SA (2)	WA (3)	SA (3)
8	$6.23 \cdot 10^{-3}$	≈ 1	≈ 1	≈ 1
35	$2.24 \cdot 10^{-10}$	$9.72 \cdot 10^{-3}$	$2.29 \cdot 10^{-7}$	≈ 1
200	≈ 0	$3.17 \cdot 10^{-12}$	≈ 0	$3.24 \cdot 10^{-9}$
2000	≈ 0	≈ 0	≈ 0	≈ 0

Table 2Values of s (obtained by inverting (2) and (3)) for $m = 1024$, $n = 4096$ and different values of γ .

γ	Lower bound on s [13]		New lower bound on s	
	WA	SA	WA	SA
10^{-2}	8	35	19	88
10^{-5}	19	87	30	140
10^{-10}	37	174	48	227

We notice that the actual number of samples asked by each node is much larger than expected, resulting in a larger sampling cost S , which increases linearly with s as follows [13]

$$S = s \left(\frac{B}{k} + [y(b-1) + yb(1-R)] \log_{bR} \frac{k}{Rt} \right),$$

where B is the block size, y is the hash size and b , introduced in Section 2.3, is the number of batched hashes in each layer. Finally, t is the number of hashes in the root and determines the header size $H = t\ell_{\mathcal{H}}$, where $\ell_{\mathcal{H}} = 256$ is the binary length of the digests.

In Table 3, we assess the sampling cost S , normalized with respect to the block size B , assuming $m = 1024$, $R = 1/4$, $k = 1024$ symbols, $B = 1$ MB, $b = 8$, $t = 256$ hashes and some different values of the ASP γ . A comparison with the optimized ASBK protocol [12], named after the original authors' initials, is also reported, for which we have considered the same block size, and codes defined over a field of size 2^{256} . As expected, the optimized ASBK protocol results in smaller sampling costs than the SPAR protocol (this also held true for the original ASBK protocol [13, Fig. 4]).

Table 3Sampling cost S normalized to the block size B for $m = 1024$, $n = 4096$ and different values of γ .

γ	Lower bound on S/B [13]		New lower bound on S/B		Lower bound on S/B [12]
	WA	SA	WA	SA	
10^{-2}	0.0233	0.1019	0.0553	0.2563	0.0278
10^{-5}	0.0533	0.2534	0.0874	0.4077	0.0358
10^{-10}	0.1078	0.5068	0.1398	0.6611	0.0435

However, it should be noticed that SPAR has the advantage of relying on a fixed header size whereas in ASBK the header size increases as the square root of the block size. Therefore,

considering the same setting, in Tables 4, 5 and 6 we have compared the total amount of downloaded data D (sampling cost plus header size) using SPAR, to that obtained using the optimized ASBK protocol. The tables also report the header size H for the optimized ASBK protocol, when $B = 1$ MB, $B = 10$ MB and $B = 100$ MB, respectively. The header size for SPAR does not depend on the block size and its value is $tl_{\mathcal{H}} = 8.192$ kB. Notice that this amount of data must be downloaded by any light node during the regular course of the protocol, independently of the malicious behaviour of some full nodes, possibly resulting in the additional download of fraud proofs.

Table 4

Total amount of downloaded data normalized to the block size $B = 1$ MB for $m = 1024$, $n = 4096$ and different values of γ .

γ	New lower bound on D/B		Lower bound on D/B [12]	H [kB] [12]
	WA	SA	-	-
10^{-2}	0.0635	0.2645	0.0454	20.411
10^{-5}	0.0956	0.4159	0.0544	
10^{-10}	0.148	0.6693	0.0639	

Table 5

Total amount of downloaded data normalized to the block size $B = 10$ MB for $m = 1024$, $n = 4096$ and different values of γ .

γ	New lower bound on D/B		Lower bound on D/B [12]	H [kB] [12]
	WA	SA	-	-
10^{-2}	0.009	0.0386	0.0099	52.244
10^{-5}	0.0137	0.0609	0.0123	
10^{-10}	0.0214	0.0983	0.0154	

Table 6

Total amount of downloaded data normalized to the block size $B = 100$ MB for $m = 1024$, $n = 4096$ and different values of γ .

γ	New lower bound on D/B		Lower bound on D/B [12]	H [kB] [12]
	WA	SA	-	-
10^{-2}	0.0012	0.0051	0.0022	158.03
10^{-5}	0.0018	0.008	0.0025	
10^{-10}	0.0028	0.013	0.0031	

We observe that, for relatively small and moderate values of the block size, despite the larger header size, the use of the ASBK protocol is preferable even if a weak adversary is taken into account. Instead, when the block size is large, SPAR is very convenient in the presence of a weak adversary, but still more costly than ASBK if the adversary is strong.

5. Conclusion

By carefully analyzing the SPAR protocol we have shown that the actual sampling cost required by the scheme, in order to achieve target security guarantees, is much larger than that initially expected. Moreover, it is shown that, in many practical scenarios, the amount of data that light nodes have to download is larger than that of other well-known schemes.

References

- [1] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. <https://arxiv.org/pdf/1905.09274.pdf>, 2019.
- [2] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. <https://arxiv.org/pdf/1809.09044.pdf>, 2019.
- [3] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In N. Borisov and C. Diaz, editors, *Financial Cryptography and Data Security, FC 2021*, volume 12675 of *Lecture Notes in Computer Science*, pages 279–298. Springer, Berlin, Heidelberg, 2021.
- [4] Matteo Bernardini, Diego Pennino, and Maurizio Pizzonia. Blockchains meet distributed hash tables: Decoupling validation from state storage. In *CEUR Workshop Proceedings, 2nd Distributed Ledger Technology Workshop (DLT)*, volume 2334, pages 43–55, Pisa, Italia, February 2019.
- [5] Steven Cao, Swanand Kadhe, and Kannan Ramchandran. CoVer: Collaborative light-node-only verification and data availability for blockchains. In *Proceedings of the 2020 IEEE International Conference on Blockchain (Blockchain 2020)*, pages 45–52, Rhodes, Greece, November 2020.
- [6] Karunakaran Murali Krishnan and Priti Shankar. Computing the stopping distance of a Tanner graph is NP-hard. *IEEE Transactions on Information Theory*, 53(6):2278–2280, 2007.
- [7] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.
- [8] Debarnab Mitra, Lev Tauz, and Lara Dolecek. Concentrated stopping set design for coded Merkle tree: Improving security against data availability attacks in blockchain systems. In *Proceedings of the International Symposium on Information Theory (ISIT 2020)*, pages 136–140, Los Angeles, CA, USA, June 2020.
- [9] Debarnab Mitra, Lev Tauz, and Lara Dolecek. Concentrated stopping set design for coded Merkle tree: Improving security against data availability attacks in blockchain systems. <https://arxiv.org/pdf/2010.07363.pdf>, 2021.
- [10] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. : <https://bitcoin.org/bitcoin.pdf>, 2008.
- [11] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008.
- [12] Paolo Santini, Giulia Rafaiani, Massimo Battaglioni, Franco Chiaraluze, and Marco Baldi.

- Optimization of a Reed-Solomon code-based protocol against blockchain data availability attacks. In *Proceedings of IEEE International Conference on Communications (ICC) 2022*, Seoul, South Korea and virtual, May 2022.
- [13] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded Merkle tree: Solving data availability attacks in blockchains. In J. Bonneau and N. Heninger, editors, *Financial Cryptography and Data Security, FC 2020*, volume 12059 of *Lecture Notes in Computer Science*, pages 114–134. Springer, Cham, 2020.
- [14] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *IEEE Access*, 8:16440–16455, 2020.

Evaluating Blockchain Systems: A Comprehensive Study of Security and Dependability Attributes

Stefano De Angelis^{1,*}, Gilberto Zanfino¹, Leonardo Aniello¹, Federico Lombardi^{1,2} and Vladimiro Sassone¹

¹University of Southampton, University Rd, Southampton SO171BJ, UK

²Conio Inc., San Francisco, California, U.S.A.

Abstract

Blockchain is the enabling technology behind decentralised, fully peer-to-peer, systems. It distributes trust across a network of autonomous entities without the need for centralised trusted authority. It is therefore easier for an attacker to add malicious nodes that remain undetected. The absence of trust results in a more vulnerable system, where adversaries may come both from the inside and outside. In this context, security guarantees become crucial to ensure blockchains' reliability and trust.

In this work, we propose a comprehensive evaluation of security attributes for blockchains. We refer to the well-established concepts of security and dependability, broadly used in distributed systems, to identify the most relevant properties for blockchains. Thus, we use such properties to evaluate five of the most prominent, platforms, such as three permissionless blockchains -Bitcoin, Ethereum 2.0, and Algorand- and two permissioned blockchains -Ethereum-private and Hyperledger Fabric. We assess security over three dimensions, i.e. the consensus, infrastructure, and smart contracts.

Keywords

Blockchain, Security and Dependability, Consensus,

1. Introduction

Blockchain replaces traditional centralised infrastructures through a distributed network of entities that collectively fulfill operations without the need of trusting each other. The advantages of decentralisation are threefold: no single point of failure, distributed trust, and a system harder to compromise [40]. However, correctness and reliability are traded for complex distributed computing procedures. The increased complexity and lack of trust lead to a more vulnerable system due to a wider attack surface. For instance, in 2021 about US\$ 1.3B got stolen in decentralised finance applications [13] by exploiting code issues related to smart contracts -programs deployed and executed on the blockchain.

Security is nowadays a paramount need for blockchains. In traditional distributed systems, security is often paired with *dependability*. Those properties include a set of attributes that

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy


*Corresponding author.

✉ s.deangelis@soton.ac.uk (S. De Angelis); g.zanfino@soton.ac.uk (G. Zanfino); l.aniello@soton.ac.uk (L. Aniello); federico.lombardi@conio.com (F. Lombardi); vsassone@soton.ac.uk (V. Sassone)

🆔 0000-0002-1168-9064 (S. De Angelis); 0000-0002-5576-3246 (G. Zanfino); 0000-0003-2886-8445 (L. Aniello); 0000-0001-6463-8722 (F. Lombardi); 0000-0002-6432-1482 (V. Sassone)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

identify the reliability, availability, confidentiality, and integrity of a system during its execution [6, 7]. In a blockchain context, where several parties exchange value via peer-to-peer transactions, it is crucial ensuring that the system remains secure and dependable thus avoiding problems like double-spending. However, blockchain systems entail several infrastructures and architectural choices such as the use of either a permissionless or permissioned network, the *consensus* protocol, and the use of *smart-contracts* for applications. As a result, assessing the security of each component might be a challenging task. In literature, some effort has been devoted to studying security in consensus protocols employed for blockchain systems [11, 38, 41]. However, a fair comparison is elusive due to several contrasting assumptions. Moreover, some works attempted to provide security evaluation of blockchains applications by assessing exploited vulnerabilities of smart contracts [28, 5, 37], however, most of these studies mainly focus on the Ethereum platform [43].

In this paper, we propose a comprehensive evaluation of blockchains' security aspects. We provide a refined definition of security and dependability referencing the traditional properties of *safety* and *liveness* and the CIA Triad - *confidentiality*, *integrity*, and *availability*. Thus we introduce two new properties, namely *profiling* and *fairness*. The former determines the ability of a blockchain to authenticate participants and define access control rules. The latter models the willingness of a system to be accessible by any participant and to process operations democratically. We evaluate those properties with respect to three dimensions, namely *consensus*, *infrastructure* and *smart contracts*. We consider five most prominent blockchain platforms, namely *Bitcoin* [32], *Ethereum 2.0* [20], *Algorand* [24] *Hyperledger Fabric* [3] and a private instance of Ethereum, called *Ethereum private* [23]. Firstly, we study the architectural models of these platforms. Then, we focus on the underlying consensus protocols, i.e. the mechanism used by the network to democratically agree on the order operations. The proposed analysis distinguishes three types of attack vectors targeting *assets*, i.e. 'computing' in PoW and 'stake' in PoS, or *network nodes*, i.e. the maximum number of subverted nodes that PoA and PBFT can tolerate. Finally, we drift the analysis to the application layer built on top of the smart-contracts capabilities of blockchains. We provide a detailed description of well-known code issues affecting smart contracts and thus evaluate how each issue impacts security.

The rest of this paper is divided as follows. Section 2 introduces the blockchain platforms we consider in our study and Section 3 describes their underlying consensus protocols, whereas Section 4 presents a collection of smart contract code issues. Then, Section 5 defines the refined security and dependability properties and the security analysis of those properties at consensus, platform, and smart contract layers. Finally, Section 6 sums up the results.

2. Blockchain Platforms

In this section, we describe the blockchain platforms considered in our analysis, namely Bitcoin, Ethereum, Algorand, Ethereum-private and Hyperledger Fabric. We briefly introduce the architectures, yet an overview of their performance and security.

Bitcoin. Bitcoin [32] is the first, open-source, permissionless blockchain born for electronic *machine-to-machine* payment without need of any central authority. Bitcoin transactions are processed in a fully decentralised manner and their ordering is guaranteed by an underlying

lottery-based (i.e. probabilistic) consensus mechanism, i.e., the PoW. Such a solution allows a consistent, immutable, and therefore trustworthy, public ledger of transactions ever made. However, in the lottery-based consensus model, valid blocks can get mined at the same time; this makes it possible to fork the blockchain in multiple valid branches. To compromise a block an attacker must control 51% of the computational power of the miners. Compromising more than 6 blocks is considered computationally infeasible, therefore a block is considered final after ≈ 6 blocks. To avoid double spending and/or avoid spending tokens not owned, Bitcoin uses the UTXO model, i.e., each transaction is composed of a list of unspent transactions indicating the balance of accounts. Besides, the sender of a transaction is charged a mining fee whose amount depends on the size of the transaction, i.e., the number of UTXO addresses used. To ensure strong (eventual) integrity Bitcoin sacrifices performance, indeed the throughput is only about 5 txn/s with a block confirmation period of about 10 minutes.

Ethereum. Ethereum [43] is the second main open-source blockchain project. The underlying idea is to make the blockchain programmable through *smart contracts*, i.e., immutable pieces of code deployed and executed autonomously on the so-called *Ethereum Virtual Machine* (EVM). Smart contracts are developed in SOLIDITY [19], a Turing-complete programming language. The first version of Ethereum is based on the PoW consensus, like Bitcoin, but with a shorter confirmation time (about 14 seconds) which increases the throughput to about 30 txn/sec. This makes Ethereum more prone to forks than Bitcoin which are similarly solved with a longest-chain rule. The PoW makes Ethereum vulnerable to 51% attacks, like Bitcoin, therefore a block is considered final after 6 blocks. Ethereum does not employ a UTXO model to manage transactions, but an *account-based* model, i.e., each account has its balance stored within the state of the ledger. Each transaction is charged according to (i) *gas price*: the amount of ETH (the Ethereum's cryptocurrency) to be paid for each computational step; (ii) *gas limit*: a scalar value representing the total amount of gas that can be consumed by the transactions in a block.

Ethereum 2.0. It is the most important update of the Ethereum protocol to cope with scalability and performance issues. Among others, it proposes two major improvements, such as the shift from PoW to a new PoS implementation called *Casper Proof of Stake*, and the implementation of *Shard Chains*. The upgrade to PoS evolves Ethereum to a more energy-efficient platform, while Shard Chains may drastically improve scalability by changing the way the blockchain is replicated across the nodes of the network. Sharding allows parallel execution, enabling the achievement of better throughputs. However, it comes at a security cost since each shard is not managed by the entire network and hence is more vulnerable.

Ethereum Private Networks. Many implementations of the Ethereum protocol can run private networks. We refer them to as *Ethereum-private*. Two of the most common Ethereum clients are *Geth* [2], the Ethereum implementation in GOLANG language, and *Parity* [4], a RUST-based implementation. Both allow the creation of a private instance, in which transactions are visible only to a subset of network participants. Ethereum clients running private networks enable the integration of pluggable lightweight consensus algorithms. These types of chains are mainly used for business-to-business private enterprise settings which require higher performance (hundreds of txn/sec) and privacy guarantees. The security of Ethereum-private does not depend on computational power, but on the number of nodes, the attacker can control. The attacker needs to control at least 1/3 of nodes, but a wrong consensus implementation may

drastically increase the probability of attack success.

Algorand. Algorand [1] is a novel permissionless blockchain platform that aims at solving the so-called blockchain trilemma, namely, scalability, decentralisation, and security. Algorand embeds a distributed computation engine, i.e., *Algorand Virtual Machine (AVM)*, that runs on every node of the network and executes smart contracts, similarly to Ethereum. Algorand's smart contracts are self-verifiable pieces of code that run on the blockchain and automatically approve or reject transactions according to a certain logic. The AVM interprets smart contracts written in an assembler-like language called TRANSACTION EXECUTION APPROVAL LANGUAGE (TEAL). The transaction model is similar to Ethereum, namely is account-based. Algorand's core innovation is its new consensus protocol, PPoS, which can reach agreement in large networks without giving up neither scalability nor security. Algorand blockchain is designed not to fork ever, transactions are considered final as soon as executed and included in a block. This makes Algorand much faster than Ethereum with a block time of about 4.5 sec and throughput of about 1000 txn/s. Compromising Algorand requires an attacker to control 1/3 of the stake.

Hyperledger Fabric. Hyperledger Fabric [3] is a permissioned blockchain platform featured by a modular architecture. The distinguishing characteristic of Fabric is that it splits the transactions ordering, i.e. the consensus process, from transactions execution, i.e. the operations on users' assets. The assets within the ledger state are represented as a collection of *key-value* pairs, and through smart contracts (called *chain-codes* in Fabric's jargon), it is possible to combine their values to carry out complex functions according to users' needs, e.g. to perform an auction. Being permissioned, Fabric offers an *authentication* layer that identifies the system entities by issuing X.509 digital certificates. Additionally, the authentication process enforces authorisation policies on the operations. Fabric introduces the concept of *channels* to represent restricted consortium networks. Transactions within a channel remain private and shared only across channel participants, enabling data isolation and confidentiality. As the operating environment is more trusted than a permissionless setting, it allows employing a lighter consensus, which results in better performances despite restricted security assumptions.

3. Blockchain Consensus Protocols

In this section, we describe the consensus underlying the blockchain platforms mentioned in the previous section, namely PoW, CPoS, PPoS, PoA and PBFT.

Proof-of-Work. The PoW is a lottery-based consensus schema consisting of computationally-intensive hashing tasks executed by some distinctive network nodes, called miners. Specifically, miners create a block by retrieving transactions from a local pool and rush looking for a random number that, if concatenated with the transactions included in a block, makes the hash of the block lower than a target number. Such target number is adjusted over time according to a desired *difficulty*. The difficulty is chosen to keep constant the *block period*, i.e., the average time required by miners to solve the puzzle. The more the global computational power of the network, the higher the difficulty, thus the lower is the target number. After solving the PoW, the miner can broadcast the corresponding block to the network for being accepted by other nodes. If accepted, all the correct nodes consider it as the latest block in the chain and start mining new blocks on top of it. Due to the probabilistic nature of the PoW, forks may happen. It

is the responsibility of the platform implementing the PoW to find a strategy to cope with forks. The standard approach used by Bitcoin and Ethereum, as mentioned in the previous section, is the longest-chain rule. Transactions may include a mining fee to incentive miners to pick that from the pool. Thus, transactions with zero or low mining fees may never be included in a block generating *starvation* [34, 32]. The verification and subsequent acceptance procedures happening in PoW make a block *persistent* unless an attacker controls the majority of the miners' hash power (the aforementioned 51% attack), which enables it to create a chain fork with modified transactions. However, being based on computational power rather than several nodes, it is not vulnerable to *sibling attacks*. Although provides strong integrity properties, besides energy inefficiency, PoW has *performance* limitation due to the intensive hashing tasks.

Casper Proof-of-Stake. The *Proof-of-Stake* (PoS) works by deterministically selecting a set of validators according to their cryptocurrency holdings, i.e. their stake. Any node committing a stake can become a validator by locking up their stake amount into a deposit. The validators propose and vote on the next block, and the weight of each validator's vote depends on the deposit amount. In Ethereum's PoS implementation, called *Casper* (CPoS) [18], each validator's turn is determined by one of the following techniques: *Chain-based PoS*: the algorithm pseudo-randomly selects a validator during each time slot to propose a block; *BFT-style PoS*: a multi-round process, in which each validator sends its vote for a specific block. At the end of the process, all (honest and online) validators permanently agree on whether or not any given block is part of the chain. Conversely to PoW, the CPoS protocol causes no waste of energy since it does not requires computational tasks to be solved, therefore, performance can be much better. However, if a validator does not follow the consensus rules, PoS applies penalties. Attacking a PoS requires an attacker to control the majority of committed staking, making it not vulnerable to *sibling attacks*. However, it is crucial not to make predictable the leader, otherwise, the attacker just needs to compromise a much smaller set of nodes that may be elected as a leader; in this case, the security drops from the ideal majority of committed staking to compromised nodes.

Pure Proof-of-Stake. The PPoS [24] is the underlying consensus of Algorand. It leverages VRF (Verifiable Random Functions) [30] to significantly decrease the high volume of exchanged messages occurring in traditional voting-based and lottery-based consensus. Specifically, PPoS works as follows: it proceeds in rounds, and for each round there are three phases: *block proposal*, *soft vote*, and *certify vote*. When a round starts, users use the VRF to select themselves as leader and committee members. In the *block proposal* phase, the leader selected by the VRF propagates a new block along with the VRF output, which proves that the account is a valid proposer. Then in the *soft vote*, a selected committee of users cast a vote on the block proposals. This phase reduces the number of proposals down to one, guaranteeing that only one block gets certified in a round. When a quorum of votes from the committee members is reached starts the *certify vote*. In this last phase, a new committee checks the validity of the block at the *soft vote* stage. Thus a new vote begins to certify the block. When a quorum is reached, the block is committed and the round terminates. Each phase is characterized by a timeout to ensure safety when partitions occur. PPoS can achieve higher throughput and lower block time than traditional PoS due to a reduced message exchange. Furthermore, the VRF makes the leader unpredictable, dismissing the possibility of having fixed validators such as in traditional PoS.

Practical Byzantine Fault Tolerance. The *PBFT* consensus protocol [12] is characterised by a

single-leader and view-change approach. The algorithm proceeds in views, for each view there exists a leader and a set of replicas. Each view executes a *three-phase commit* protocol where replicas exchange messages to reach the total order of transactions. In case of misbehaving leaders, all the correct replicas run a view change operation which starts a new view and elects a new leader. In an eventually-synchronous network, where messages are delayed and network partitions may happen, the PBFT consensus protocol guarantees strong consistency provided that $f < N/3$, with f malicious nodes and N replicas. PBFT has been proven to be optimal with $N \geq 3f + 1$ nodes [12]. PBFT is vulnerable to sibling attacks though since it cannot distinguish if an attacker is falsely impersonating multiple nodes.

Proof-of-Authority. The *PoA* has been proposed as part of the Ethereum consortium for private networks and implemented with two protocols called *AuRa* and *Clique* [35, 39]. PoA relies on a set of trusted authorities running the consensus. Consensus in PoA relies on a *leader rotation* schema, which distributes the responsibility of block creation among the authorities [9, 22]. Time is divided into *steps*. In each step, an authority is elected as the proposer. The way authorities are elected differs in the two consensus protocols. AuRa proposes a deterministic function based on UNIX times, which requires strong synchronisation assumptions. Conversely, Clique computes leaders according to the number of the next block on the blockchain. The PoA is a hybrid consensus protocol between the lottery-based and voting-based approaches. PoA protocols guarantee eventual consensus on transactions. Indeed, the lightweight leader election may lead to forks that eventually get resolved. Consequently, PoA cannot achieve instant finality but this is delayed in time. According to the concept of the longest chain, a block in PoA is considered final when a majority of further blocks have been proposed, under the assumption that blocks are proposed at a constant rate [35]. These algorithms are vulnerable to sibling attacks, and thus cannot be used in permissionless settings.

4. Smart Contracts Issues

Beyond secure-by-design due to consensus algorithms, a prominent security role is played by smart contracts. In this section, we evaluate potential issues that affect the smart contract-enabled platforms, such as Ethereum, Algornad, and Hyperledger Fabric, thus we consider possible preventions/mitigation methods.

(I_1) *Reentrancy*. This vulnerability occurs when a caller contract invokes a function of an external callee contract. Specifically, a malicious actor can call back from the callee contract funds withdraw function of the caller contract, *i.e.*, *reentrancy*, before the execution of the caller triggering an infinite loop of calls. This allows the attacker to bypass the validity checks of the caller and iterate infinitely. Ethereum is vulnerable to reentrancy and its exploitation may lead to indefinite withdrawal calls. Two reasons cause this vulnerability [36]: (i) validity checks are handled by state variables that are not updated until other transactions terminate, (ii) no gas limit is required when handling interactions between external smart contracts. Prevention methods consists in (i) update the state variables before calling external contracts; (ii) introduce a *mutex lock* [19] in the contract state so that only the owner can update such state. Similarly, Hyperledger Fabric suffers reentrancy since chaincodes-to-chaincodes are allowed with no limitations inter-channel. Fabric mitigates such issues through a timeout,

however, it is important to note that reentrancy has a limited impact on private settings since no cryptocurrencies are involved. Conversely, Algorand does not suffer reentrancy since contract-to-contract calls are allowed one way only, thus if smart contract A calls a smart contract B, the latter cannot call back A.

(I₂) *Integer overflow and underflow*. This vulnerability occurs when a function computes an arithmetic operation that falls outside a specific datatype. A prominent role is played by the programming language. Furthermore, some protections there exist both natively or through an external library. Ethereum does not provide native prevention for smart contracts, but some recommendation has been defined, such as (i) using *SafeMath* library [33] to check on underflow/overflow, (ii) using *Mythril* library [15] to check the security of EVM bytecode before its execution. Algorand does not need any prevention library as TEAL natively copes with under/overflow issues. Hyperledger Fabric, being based on golang makes us of the native under/overflow management or common libraries such as *overflow*.

(I₃) *Frozen Token*. This vulnerability causes users' funds deposited in the contract account to be locked and impossible to withdraw back, effectively freezing them into the contract. Both Ethereum and Algorand are vulnerable to such an issue. The causes of this vulnerability are twofold: (i) the deposit contract account does not provide any function to spend funds using a function from an external contract as a library, (ii) the callee contract function (`selfdestruct` for Ethereum, `clearState` for Algorand) is executed without checks. Prevention method [14, 37]: a deposit contract shall assure that the mission-critical functions or money-spending functions are not outsourced to external contracts. Hyperledger Fabric is not vulnerable since no cryptocurrencies are involved.

(I₄) *DoS with unexpected revert*. This issue occurs if the execution of a transaction is reverted due to a thrown error or a malicious callee contract that deliberately interrupts the execution. Ethereum, Algorand and Hyperledger Fabric are vulnerable. For Ethereum, a best practice to mitigate the issue regards the transaction sender to provide to the callee a certain amount as a reward for the execution of a transaction so that the callee is not incentivized to revert. Algorand does not have mitigation in place since no reward fees are available. Fabric, similarly, does not have solutions to avoid it, due to the absence of cryptocurrencies.

(I₅) *DoS with GasLimit*. This vulnerability causes transactions to be aborted due to exceeding the gas limit. This affects only Ethereum due to the presence of gas. To mitigate this issue the contracts should not execute loops on accounts accessible data structures. Loops should be controlled, such that the execution always terminates, even when transactions are aborted.

(I₆) *Insufficient signature information*. This vulnerability causes a digital signature to be valid for multiple transactions. This happens when a sender uses a *proxy* contract [14, 37] as a deposit for one or more authorised receivers. An authorised receiver owns a digitally signed message delivered off-chain from the sender. The receiver withdraws funds from the proxy, which must verify the validity of the digital signature. If the signature is malformed (missing nonces, or proxy contract address), a malicious receiver can reuse the signature to reply to the withdrawal transaction and drain the proxy balance. Prevention method: The contract shall check the address and the nonce within digitally signed messages. Both Ethereum and Algorand are vulnerable to this issue, while Hyperledger Fabric is not since it authenticates network nodes.

(I₇) *Generating randomness*. This vulnerability concerns smart contracts using pseudorandom number generators (PRNG) to create random numbers for application-specific use cases. Specifi-

cally, this vulnerability affects methods using random numbers created by a PRNG, in which the base seed of the generator is a parameter controlled by miners, e.g. Solidity's `block.number`, `block.difficulty`. A malicious miner can manipulate the PRNG to generate an output that is advantageous for itself. Ethereum, Algorand and Hyperledger Fabric are all affected by this issue. For mitigation, mining variables should not be used in control-flow decisions. Therefore, off-chain approaches to PRNG should be used, such as the use of oracles.

(*I₈*) *Block Timestamp manipulation.* This vulnerability affects smart contracts that use `timestamp` parameter in the control-flow (e.g. for periodic payments) or as source of randomness [14]. As miners can control this parameter, they could adjust that value to change the logic of functions, and thus take profit. Ethereum is vulnerable to such issues and a prevention method consists in avoiding the parameter in any control-flow decision logic. Algorand is not vulnerable since it employs a maximum timestamp offset of 20 seconds between two blocks. Similarly, Hyperledger Fabric has no constant block time.

(*I₉*) *Transaction ordering dependence.* This vulnerability is caused by a malicious manipulation of the transaction priority mechanism used in Ethereum. Gas is usually used to prioritise the execution of certain transactions over others [43]. However, malicious miners can alter this procedure and always prioritise their transactions regardless of the gas price, hence manipulating the global state of the blockchain in its favor [37]. Mitigation method: hide the gas price from transactions using cryptographic committees or guard conditions [14]. Algorand and Hyperledger Fabric are not affected by this issue since they do not use gas.

(*I₁₀*) *Under-priced opcodes.* This vulnerability is caused by under-priced opcodes that can be executed at low cost and that consume a large number of resources. Misuse of the opcodes, or a malicious actor, might trigger several invocations of these opcodes wasting the majority of resources. This vulnerability regards Ethereum and it is caused by misconfigured gas price parameters [14]. The Ethereum protocol has been upgraded to limit opcodes under-pricing. Algorand is not affected since the cost is set 1 to all opcodes with a limit of 700 per application. Hyperledger is not affected due to the nature of private blockchains' costless computation.

(*I₁₁*) *Token lost to orphan address.* This vulnerability is caused by a lack of validation checks on payment transactions. Ethereum only checks the recipient's address format but not if such an address is valid nor if it exists. If a user sends money to non-existing addresses, Ethereum automatically creates a new *orphan* address [5]. An orphan address is neither an EOA nor a contract address, hence the user can't move the money which is indeed lost. Algorand has the very same issue, with a small client-side check of existence as mitigation implemented in all the official clients and SDKs. The only effective prevention method at the time of writing is to manually assure the correctness of the recipient's address [14]. Hyperledger nodes are instead authenticated, thus it is not affected.

(*I₁₂*) *Short address.* This vulnerability affects only Ethereum and it is caused by the EVM missing validation check on addresses. Recall that inputs are expressed as an ordered set of bytes, in which the first four bytes identify the callee's function, then other inputs are concatenated in chunks of 32 bytes. In case of arguments with fewer bytes, EVM auto-pads with zeros to generate the 32 bytes chunk. An attacker could manipulate this process to execute malicious actions. For instance, if we consider the `transfer(address addr, uint tokens)` function and a bad formatted `addr` with one missing byte, the auto-pad adds extra zeros at the end of `addr`, and consequently increases the number of tokens to transfer [14]. To prevent that,

write functions that validate the length of the transaction’s inputs. Algorand has prevention by design, it does not add extra zeros as padding and the transaction fails in case of a short address. Hyperledger Fabric, as above, is not affected due to the authentication of nodes.

(*I*₁₃) *Erroneous visibility*. This vulnerability takes advantage of Ethereum’s public nature. Transactions (including data, balances and contract codes) are visible to any user. However, Solidity provides four types of visibility to restrict the access to a contract’s functions, namely `public` open to everyone, `external` only externally from the contract, `internal` only internally (the contract and its related contracts) , and `private` only within the contract. By default, Solidity assigns the type `public` to functions, hence in case of erroneous visibility configuration, an attacker might be able to call the function from the external [14]. To avoid this, with Solidity 0.5.0 and above, it is mandatory to specify the function visibility. Algorand conversely has all functions public only. Hyperledger can hide data in several methods such as Trusted Execution Environment with Intel SGX, channels [8].

(*I*₁₄) *Unprotected suicide*. In Ethereum, Solidity contracts can be killed or deleted using the `suicide` or `self-destruct` methods. Usually, only the contract’s owner, or authorised external users, can invoke these functions. However, there might be cases in which the owner is not verified by the functions, or the owner itself is malicious, in that case, an attacker can invoke one of these methods and kill the contract. The very same situation happens with Algorand through the `ClearState` function. Prevention method: enhance security checks with, permissions mechanisms, to assure that the `suicide/self-destruct` and `ClearState` calls are approved by different parties. Hyperledger Fabric is not affected.

(*I*₁₅) *Unrequested Token*. In Ethereum ERC-20 tokens can be sent to an arbitrary address. This is used as a common phishing technique where a malicious actor creates an ERC-20 token and sends some amount to random addresses. The token is designed with a `sell` smart contract function which drains the wallet. When a phished user attaches his wallet to the application controlling this contract, the user unintentionally authorises the smart contract to steal his funds. Algorand mitigates this issue via opt-ins. Hyperledger Fabric is not affected since no cryptocurrencies are involved.

5. Evaluation of Security Properties for Blockchain

In this section we introduce security and dependability attributes for blockchains and we evaluate them over three dimensions, i.e., *consensus*, *infrastructure* and *smart contracts*. The proposed analysis follows a qualitative evaluation that takes into account the descriptions of the platforms and protocols detailed in the previous section. Therefore, the analysis considers how the identified properties are met in each system. The methodology used assumes the deployment of n nodes responsible for consensus, which communicate over an eventually synchronous network [17]. Such a model realistically describes the Internet network, where messages can be arbitrarily delayed, but eventually delivered within a fixed time-bound.

A distributed protocol is considered if satisfies *safety* and *liveness* properties [11]. However, in a blockchain context, the traditional definition of such properties does not straightforwardly apply. For instance, transactions are asynchronously committed by the network after the execution of a consensus protocol. For this reason, safety and liveness must be refined in order

Table 1
Security evaluation of blockchain consensus protocols

	<i>PoW</i>	<i>CPoS</i>	<i>PPoS</i>	<i>PBFT</i>	<i>PoA</i>
<i>persistence</i>	eventual	eventual	yes	yes	eventual
<i>termination</i>	yes	yes	eventual	$n \geq 2f + 1$	eventual
<i>validator fairness</i>	hw dependent	stake dependent	stake dependent	yes	yes

to explicitly reflect the behavior of a blockchain system. To cope with this limitation, we start from the traditional definitions of safety and liveness [11, 6, 7] to introduce two novel properties, namely *persistence* and *termination*. We also provide a new metric for blockchains, i.e. the *fairness* property. Following seminar work by Francez [21], we distinguish two aspects of the fairness: *validator fairness*, for consensus protocols and *client fairness*, for infrastructures. The novel security attributes for consensus in blockchain are thus summarised as follows:

1. *Persistence*: nothing wrong happens during the consensus execution; unwanted executions must be prevented. When an honest node accepts a transaction, then all the other honest nodes will make the same decision, which is irreversible. If persistence is violated after a certain threshold (i.e. confirmation time), it will never be satisfied again. Persistence is also referred to as *finality* [41].
2. *Termination*: ensures that a protocol makes progress towards an end, hence transactions correctly terminate, i.e. the block including those transactions reaches persistence.
3. *Validator fairness*: in a blockchain, the consensus mechanism is fair if any honest node can be potentially selected to the set of nodes that will participate in the agreement to select the next block.

Table 1 summarises the consensus resilience of the four algorithms acting under the adversarial model presented with our methodology. Firstly, we observe that PoW and CPoS enjoy strong termination thanks to their probabilistic leader election based on mining, and staking. Transactions are guaranteed to be added to the blockchain as soon as the mining proceeds, or there is a majority of stakeholders. Conversely, probability in leader election affects persistence, because of the possibility of having forks. However, such forks will eventually be resolved, according to the specifics of the platform. For this reason, persistence must be classified as *eventual*. Conversely, the PPoS protocol does not allow forks, and blocks are instantly finalised, prioritising persistence over termination [24]. Indeed, the PPoS allows stalls in case of misbehaviors or network issues. However, PPoS’ introduces a recovery mechanism to ensure termination, so we classify that as *eventual*. Both PoS protocols ensure security until a majority of the stake remains in honest hands. If this condition is not verified, such systems can be easily compromised. Validators with the majority of the stake can determine the next blocks straightforwardly. This behavior is reflected with the validator fairness property in Table 1. Differently in PoW, such property is strongly related to hardware capabilities. Miners with outstanding computational power will have more chances to produce blocks.

Moving to permissioned blockchains, these systems rely on a higher level of trust, due to the presence of node authentication. The consensus protocols used in this context are either classical

Table 2

Security evaluation of blockchain platforms

	<i>Bitcoin</i>	<i>Ethereum 2.0</i>	<i>Algorand</i>	<i>Hyperledger</i>	<i>Ethereum-private</i>
<i>confidentiality</i>	no	no	co-chains	channels	private txs
<i>integrity</i>	majority of hash power	majority of stake	majority of stake	up to $3f + 1$	eventual
<i>availability</i>	yes	yes	yes	up to $2f + 1$	eventual
<i>accountability</i>	partial	partial	partial	yes	yes
<i>authorisation</i>	no	no	no	yes	yes
<i>client fairness</i>	no	no	yes	yes	yes

voting-based ones, such as PBFT, or hybrid, as for PoA. PBFT has been broadly demonstrated to guarantee persistency in the eventually synchronous model, as long as there are $n \geq 3f + 1$ active nodes in the network. Whereas as soon as $n \geq 2f + 1$ are honest, termination can be also guaranteed [16]. In PoA, persistency is only *eventually* guaranteed, because PoAs are open to forks. Termination is instead eventual, according to the number of honest validators. As long as a majority of validators are active, termination is guaranteed, otherwise, the protocol stalls and transactions are not finalised [16]. Finally, PoA’s validator fairness is guaranteed, since every honest validator has the same chance of being elected as a leader as the others.

Turning to the security evaluation of blockchain platforms, we define six attributes based on the traditional definitions of security and dependability [6], i.e. the *CIA triad* and the user’s *profiling*. We identify the relevance of the properties of *accountability* and *authorisation*. Such properties lead to fairness constraints which can be used to detect misbehaving participants. Hence, our suggested attributes are:

1. *Confidentiality*: the possibility to keep some transactions confidential; absence of unauthorised leaking of sensitive information owned by one or more nodes;
2. *Integrity*: absence of improper alterations of the blockchain data from unauthorised users;
3. *Availability*: the ability of the system to run correct services without interruptions;
4. *Authorisation*: the ability of the system to specify access rights and privileges to resources and to define permission roles for participants;
5. *Accountability*: the ability of the system to trace back the operations and the behaviour of a certain user identity/physical entity;
6. *Client fairness*: the willingness of the system to democratically accept transactions from any client without any preference.

Table 2, shows our analysis with the aforementioned six attributes. We observed that the integrity of Bitcoin, Ethereum 2.0, and Algorand is strongly linked to where hash power and stake lie. Indeed, an attacker owing the majority of the hash power (or stake), could break the consensus protocol as already mentioned, hence maliciously injecting a fork with a subverted chain [10]. In contrast, in Hyperledger and Ethereum-private, the integrity property is strongly tied to the persistency property of their underlying consensus algorithms. Fabric employs PBFT, which ensures persistency under the assumption of $n \geq 3f + 1$, while Ethereum-private adopts

Table 3
Security issues and native resistance/mitigation

Issue ID	Security Issues	Native Resistance/Mitigation		
		Ethereum	Algorand	HL Fabric
I_1	CI + authorisation		✓	
I_2	CI + authorisation		✓	✓
I_3	A + authorisation			✓
I_4	A			
I_5	A		✓	✓
I_6	CI + authorisation			✓
I_7	I + authorisation			
I_8	IA		✓	✓
I_9	IA + accountability		✓	✓
I_{10}	A		✓	✓
I_{11}	I			✓
I_{12}	I + authorisation		✓	✓
I_{13}	CI + authorisation	✓		✓
I_{14}	all			✓
I_{15}	authorisation		✓	✓

PoA algorithms, which can only guarantee eventual persistency. Despite strong availability, the full replication of the blockchain in the Bitcoin, Ethereum 2.0, and Algorand platforms leads to a lack of confidentiality due to the public nature of the information stored on these blockchains [25]. However, if for Bitcoin and Ethereum 2.0 there is no way to mitigate this issue, Algorand recently designed a solution which combines the public, permissionless network with several private networks interconnected, *a.k.a.*, *Co-Chains* [29]. Contrarily, confidentiality in both Hyperledger and Ethereum-private can be guaranteed through the use of channels and private transactions, respectively. Hyperledger Fabric and Ethereum-private can enforce the so-called profiling properties of authorisation and accounting. This is because nodes are authenticated. Authorisation is guaranteed by managing the permission of each node. Accountability is achieved by tracing the interaction of nodes with the blockchain [26]. This is not so in public permissionless blockchains like Bitcoin, Ethereum 2.0, and Algorand where identities are pseudo-anonymous [25] and users are not authenticated. However, although actions are difficult to attributable to specific entities, it is possible to analyze the behaviour of specific accounts. Therefore, the public, permissionless nature of these blockchains ensures that anyone can access the history of transactions and trace behavioural patterns. We deduce that permissionless blockchain offer *partial* accountability [27, 31]. On the other hand, being these systems public and decentralised, authorisation is not provided. Lastly, we evaluate the property of client fairness. Permissioned blockchain benefits from fairness guarantees in that each client's transactions are processed without any preference or priority. On the contrary, the execution of transactions in Bitcoin and Ethereum 2.0 is costly (either hardware or staking), hence making incentive mechanisms for miners and validators necessary. Low-rewards transactions may be stalled forever waiting to be processed [42]. Incentives mechanisms for permissionless blockchain, like Bitcoin and Ethereum 2.0, lead therefore to a lack of client fairness. Differently,

in the Algorand blockchain, every transaction counts the same, and there is no such mechanism. Everything in Algorand is handled by PPoS cryptography and the computation of VRFs. This allows Algorand to have very low transaction fees, which are thus distributed to rewards accounts for the users, and to ensure client fairness.

We conclude our analysis with smart contracts. Table 3 illustrates a classification of the *CIA triad* and *profiling* security properties against the issues described in Section 4. From the table emerges that most of the smart contract issues may cause violation of confidentiality, integrity, and authorisation properties. The platform that shows better resilience results in Hyperledger Fabric and this is clearly due to its permissioned nature. Among the public blockchain instead, Algorand outperforms Ethereum for many issues. This is due mainly to the usage of a constant fee for transactions and opcodes conversely to Ethereum which is based on gas with a different amount for opcodes. Also, the programming language used plays a key role. Both Hyperledger Fabric and Algorand use languages that give some primitive control to avoid issues, such as control against under/overflow concerning Ethereum. Finally, some design choice related to the management of asset and smart contract calls makes Algorand more secure than Ethereum. For instance, *reentrancy* (I_1) in Algorand cannot happen by design, and tokens require to be opted-in before they can be received (I_{15}). The Ethereum naive solutions and lack of controls make it the worst in terms of security. The only situation where Ethereum outperforms Algorand is for *erroneous visibility* (I_{13}), indeed it allows to build private functions within a smart contract, while Algorand does not.

6. Conclusion

In this paper, we studied the security aspects of modern blockchain systems. We defined the security and dependability attributes which are relevant in the context of a blockchain. Thus, we analysed how five blockchain platforms, namely Bitcoin, Ethereum (with its variants Ethereum 2.0 and private chains), Algorand, and Hyperledger Fabric, guarantee those attributes. The analysis we proposed is divided into three dimensions, i.e. consensus, infrastructure, and smart contracts. Firstly, we highlighted the infrastructures' characteristics and how they differ in terms of performance (infrastructure). Then, we described their built-in consensus protocols, respectively, PoW, Casper PoS, PoA, Pure PoS and PBFT, analysing the approaches they adopt to order transactions and create blocks in a Byzantine, eventually synchronous, network model (consensus). Then, we listed smart contract issues evaluating whether blockchains are vulnerable and their native resistance/mitigations (smart contracts).

From our study emerged that permissioned blockchains like Hyperledger Fabric and Ethereum-private can guarantee fairness and confidentiality while providing accountability and authorisation. However, these platforms require strong assumptions on the underlying network and the number of possibly subverted nodes to also ensure integrity and availability. This is also reflected in the consensus protocol they adopt, specifically PBFT guarantees persistency at the cost of eventual termination, whereas in PoAs, both properties are ensured only eventually. Conversely, permissionless platforms such as Bitcoin, Ethereum 2.0, and Algorand offer better integrity and availability, despite failing on profiling, confidentiality, and client fairness properties. Finally, by studying smart contract issues we observed that Ethereum is the most

vulnerable smart contract platform compared to Algorand and Hyperledger Fabric.

References

- [1] Algorand. <https://www.algorand.com>.
- [2] Go ethereum - geth. <https://geth.ethereum.org>.
- [3] Hyperledger fabric. <https://www.hyperledger.org/use/fabric>.
- [4] Parity ethereum. <https://www.parity.io>, 2018.
- [5] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *POST - 6th International Conference, Proceedings*, volume 10204 of *Lecture Notes in Computer Science*, pages 164–186. Springer, 2017.
- [6] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dep. Secur. Comput.*, 2004.
- [7] Algirdas Avizienis, Vytautas U, Jean-claude Laprie, and Brian Randell. Fundamental concepts of dependability. 2001.
- [8] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Dev*, 2019.
- [9] BitFury Group and Jeff Garzik. Public versus private blockchains part 1: Permissioned blockchains. *White Paper*, 2015.
- [10] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE, 2015.
- [11] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild. *CoRR*, abs/1707.01873, 2017.
- [12] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.
- [13] CertiK. The state of defi security - 2021. <https://certik-2.hubspotpagebuilder.com/the-state-of-defi-security-2021>.
- [14] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Comput. Surv.*, 2020.
- [15] ConsenSys. Mythrill. <https://github.com/ConsenSys/mythrill>.
- [16] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. In *Proceedings of ITASEC*, 2018.
- [17] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
- [18] Ethereum. Proof-of-stake. <https://eth.wiki/en/concepts/proof-of-stake-faqs>.
- [19] Ethereum. Solidity. <https://soliditylang.org>.
- [20] Ethereum. Vision of ethereum2. <https://ethereum.org/en/upgrades/vision/>.
- [21] Nissim Francez. *Fairness*. Springer-Verlag, Berlin, Heidelberg, 1986.
- [22] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri,

- and Vladimiro Sassone. Blockchain-based database to ensure data integrity in cloud computing environments. In *ITA-SEC*, volume 1816. CEUR-WS.org, 2017.
- [23] Geth. Ethereum. <https://geth.ethereum.org/docs/interface/private-network>.
- [24] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.
- [25] Ryan Henry, Amir Herzberg, and Aniket Kate. Blockchain access privacy: Challenges and directions. *IEEE Security Privacy*, 16(4):38–45, 2018.
- [26] Maurice Herlihy and Mark Moir. Enhancing accountability and trust in distributed ledgers. *CoRR*, 2016.
- [27] Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Trans. Inf. Syst. Secur.*, 18(1):2:1–2:32, May 2015.
- [28] Alexander Mense and Markus Flatscher. Security vulnerabilities in ethereum smart contracts. In *Proceedings of the 20th IIWBAS*, 2018.
- [29] Silvio Micali. Algorand co-chains. <https://www.algorand.com/resources/blog/algorand-co-chains>.
- [30] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science*, pages 120–130. IEEE, 1999.
- [31] Malte Möser. Anonymity of bitcoin transactions an analysis of mixing services, 2013.
- [32] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>, 2008.
- [33] OpenZeppelin. Solidity safemath library. <https://docs.openzeppelin.com/>.
- [34] White Paper. Incentive mechanisms for securing the bitcoin blockchain white paper. 2015.
- [35] Parity. Aura - authority round consensus protocol. <https://wiki.parity.io/Aura>.
- [36] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. Sereum: Protecting existing smart contracts against re-entrancy attacks. *CoRR*, abs/1812.05934, 2018.
- [37] Noama Fatima Samreen and Manar H. Alalfi. A survey of security vulnerabilities in ethereum smart contracts. *CoRR*, abs/2105.06974, 2021.
- [38] Bano Shehar, Sonnino Alberto, Al-Bassam Mustafa, Azouvi Sarah, McCorry Patrick, Meiklejohn Sarah, and Danezis George. Sok: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019.
- [39] Péter Szilágyi. Clique - ethereum proof-of-authority consensus protocol. <https://github.com/ethereum/EIPs/issues/225>.
- [40] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *Proceedings on Privacy Enhancing Technologies*, 2017:404 – 426, 2017.
- [41] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In Jan Camenisch and Doğan Kesdoğan, editors, *Open Problems in Network Security*, 2016.
- [42] Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. On availability for blockchain-based systems. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pages 64–73. IEEE, 2017.
- [43] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014.

A Study on Diem Distributed Ledger Technology

G. Antonio Pierro^{1,*}, Roberto Tonelli^{1,†}

¹*Dep. of Mathematics and Computer Science of University of Cagliari, Palazzo Delle Scienze, Via Ospedale, 72, 09124 Cagliari CA, Italy*

Abstract

The paper analyzes the Diem Distributed Ledger Technology (DLT). First, the paper presents a general overview of the Diem project from a technical point of view. Second, it presents a study that aims to collect and analyze data from the Diem blockchain, in order to verify some properties declared in the technical paper. For instance, a relevant property of the Diem blockchain is its transactions' throughput, i.e. the rate at which valid transactions are committed into a block by the Diem blockchain in a one-second interval of time (transactions per seconds, TPS) and the interval of time for a transaction to be confirmed. The data were collected over a period of three months (January 1 - March 31, 2022) and made available on a GitHub repository.

The results of the data analysis show that the average transactions' throughput is about 60 TPS and the waiting time is on average 1 minute and 40 seconds. Moreover, the paper sheds light on some Diem features that are unique when compared to similar blockchains, such as Ethereum. Some of these unique features are the consensus mechanism based on the BFT consensus protocols (Byzantine Fault Tolerance, 2017), its accounting system based on a hierarchical model and its programming language, Move, used to code smart contracts. The analysis will provide a better understanding of the Diem blockchain's features.

Keywords

Blockchain, Virtual Asset Service Providers (VASPs), UTXO Model, Account Model, Move programming language

1. Introduction

During the last decade, many distributed payment systems have emerged as an alternative to centralized banking. The Diem Distributed Ledger Technology (DLT), initially called "Libra" and renamed "Diem" in December 2020, was designed and proposed by the Diem Association, a non-profit organization headquartered in Geneva, Switzerland. When Diem was introduced by Facebook in 2019, the Diem Association aimed to be a competitor in the field of payment systems, by introducing the Diem blockchain, i.e. a cryptographic payment system where each party is clearly identified and every transaction is authenticated, authorized, validated and

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

*Corresponding author.

†These authors contributed equally.


†These authors contributed equally.

✉ antonio.pierro@gmail.com (G. A. Pierro); roberto.tonelli@unica.it (R. Tonelli)

🌐 <https://www.agile-group.org/category/persona/> (G. A. Pierro); <https://www.agile-group.org/roberto-tonelli/> (R. Tonelli)

🆔 0000-0002-3805-7964 (G. A. Pierro); 0000-0002-9090-7698 (R. Tonelli)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

📄  CEUR Workshop Proceedings (CEUR-WS.org)

tracked. Moreover, Libra, the Diem DLT cryptocurrency, would have the ability to maintain a stable value relative to a particular fiat currency [24]. According to the technical paper at launch, the goal was to support at least 100 validators, able to process 1000 payment transactions per second. A “validator” is the term used to describe a node of the network that helps verify and propose new blocks of transaction data [7, 17].

Diem was a blockchain payment system based on an account model with users, roles and rights, where only pre-authorized computers can access and finalize transactions. This set of pre-authorized computers participate in a consensus mechanism based on the BFT (Byzantine Fault Tolerance) consensus protocols [6, 11]. Compared to other cryptocurrencies, such as Ethereum and Bitcoin, Diem has the following features:

- The withdrawal capacity i.e. the possibility to delegate the authorization to spend to a different account.
- The Diem BFT consensus protocol, i.e. a consensus mechanism where a group of authorized validators creates, verifies, and certifies the new blocks of transactions.
- An off-chain collateral system where the underlying assets are stored with an escrow service.
- The accounting system based on a hierarchical model.
- The Move programming language is used to code smart contracts and, unlike other programming languages used to code smart contracts, it integrates resources at the type level.

The online payment market economically remains massive, which suggests enormous profit opportunities for early actors on the market. Although the WhatsApp pay attempt did not reach the expected success, Facebook came back in 2019 with Libra, then called Diem, a new project that shares similarities with the previous idea. At the same time, Facebook made it clear that it did not intend to stop at the initial 28 members, which included Paypal, Shopify, Uber, eBay, and Vodafone. They were instead planning to expand the Association to over hundred members in the upcoming years.

Table 1 shows the seven largest blockchain platforms sorted by Market Capitalization and compared to the Diem DLT. The columns of the table reports some characteristics of the blockchains, such as the presence of a stable coin and smart contract support. Stable coins are cryptocurrencies which can maintain a stable price in relation to fiat currency [5]. A smart contract is a self-executing computer program that uses the blockchain to store the contract’s terms [39]. When the Diem DLT was operational, it supported a stable currency and the capability to deploy and execute smart contracts. Then, Meta, formerly of Facebook, stopped the project in January 2022.

Recently, the former Meta employees decided to continue the Diem proposal and they renamed the project to Aptos [14]. Aptos has many features in common with the blockchain Diem (<https://github.com/aptos-labs>). Some of these features are the possibility to deploy smart contracts written in the Move programming language and the possibility to build higher-level applications and protocols on top of the underlying Aptos blockchain. The Aptos’ development network (devnet) is operational since March 2022 and it is possible to monitor their transactions via a blockchain explorer named Aptos Explorer (<https://explorer.devnet.aptos.dev/>) According

to the former Meta employees, the Aptos main network (mainnet) is planned to be launched in the last trimester of 2022.

Table 1

Largest blockchain platform by Market Capitalization vs Diem blockchain

	Market Cap (Billion USD)	symbol	Permissionless?	Stablecoin?	Smart Contract?
Bitcoin	771	BTC	Yes	No	No
Ethereum	362	ETH	Yes	No	Yes
Binance	65	BNB	Yes	No	No
USD Coin	50	USDC	Yes	Yes	No
Solana	35	SOL	Yes	No	Yes
Diem	-	DIEM	No	Yes	Yes

The first technical paper specified that any interests gained with the investments of the Diem Association reserve fund, which will be composed mainly of short-term government bonds will be used to cover the costs of the system, ensure low transaction fees, and pay dividends to investors who provided capital to jumpstart the ecosystem[37].

The second version of Diem DLT was introduced with an update on the technical paper in April 2020 [35]. Many popular crypto payment systems struggle to maintain a high transaction throughput with a low transaction latency. According to the technical paper, Diem attempts to solve this with the adoption of Diem Byzantine Fault Tolerance (Diem BFT) consensus protocol. Diem BFT facilitates agreement among all validator nodes on the ordering of transactions while achieving good transaction throughput and low transaction latency when scaling in the number of validator nodes. The Diem BFT fault-tolerant model remains safe when at most one-third of the nodes are faulty.

2. Background

Nowadays, there are a large number of blockchain platforms, each with its own characteristics and design decisions [1, 33]. For instance, some platforms are designed specifically to support rich and complex smart contracts (e.g., Ethereum and Solana), while others are designed to act as a bridge between digital and fiat currencies (e.g., Tether and USDC). We list the most ten popular blockchain platforms and their characteristics vs the Diem blockchain, as depicted in Table 1.

This section describes information peculiar to the Diem blockchain such as the Move programming language 2.1 used to write smart contracts, the Proof-of-Authority (PoA) consensus algorithm and the accounting used by the Diem blockchain.

2.1. The Move Programming Language

A Smart contract is a piece of executable code that run on the blockchain to facilitate, execute, and enforce an agreement between untrustworthy parties without the involvement of a trusted third-party [23, 26]. Smart contracts have the ability to convert paper contracts into digital contracts [12, 18]. Compared to traditional contracts, smart contracts enabled users to codify their

agreements and trust relations by providing automated transactions without the supervision of a central authority [23]. In order to prevent contract tampering, smart contracts are copied to each node of the blockchain network [3, 38]. By enabling the execution of the operations by computers and services provided by blockchain platforms, human error could be reduced to avoid disputes regarding such contracts [23].

The blockchain Ethereum popularized the term by being the first public blockchain to provide a Turing complete smart contract language [41, 16]. The goal of Ethereum is to provide a world computer for which anyone can build and deploy blockchain-based applications, often referred to as Decentralized Applications (DAPPS) [27].

As well as the Ethereum blockchain, also the blockchain Diem supports smart contracts written in a different programming language which name is Move. Move is a programming language based on Rust that was created by Facebook for developing customizable transaction logic and smart contracts for the Libra digital currency. Every transaction submitted to the Libra blockchain uses a transaction script written in Move to encode its logic [4].

The key feature of Move is the ability to define customized resource types. This customized resource type supports all the operations generally available to other entities. This means the Move programming language supports passing Resource as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures. For this reason Move can be defined as a language where Resources are first-class citizen.

Resources in the blockchain system are important because they provide scarcity protections: they can only ever be moved between program storage locations, never implicitly copied or deleted. The Move type system provides static enforcement of these security measures, but allows programmers to define custom resource types.

By integrating resources at the type level rather than supporting a single type of resource value (eg, Ether), the Move programming language provides programmers with the security measures they need while remaining independent of the blockchain. Any developer can define and use custom resources, without the additional re-implementation process required by ERC20 (Ethereum Request for Comment, Proposition 20) and other libraries. To protect critical resource operations from untrusted code, Move encapsulates the fields of each resource in a corresponding form. Modules are similar to smart contracts: they contain the types and procedures for creating, updating, and destroying the assets they contain. They also provide an abstraction of critical data: fields of a resource type declared within a form are protected by any other form, and operations on that resource must only be performed within its form.

2.2. Consensus Model - Proof of Authority Consensus

Consensus makes it possible for a decentralized network of computers to agree upon and share the state of the system [1]. The consensus is critical in ensuring participants can trust the transactions processed on the blockchain even when they may not trust each other [10]. Before Bitcoin, it was impossible to electronically transfer digital money without relying on a centralized authority to manage the state of the system.

Nowadays, public blockchains such as Bitcoin or Ethereum, allow anyone to participate in the consensus process as a miner. Miners compete (or effectively vote) to add new transactions

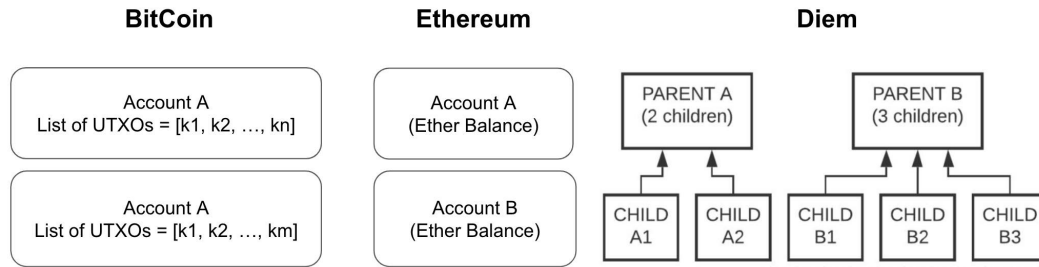


Figure 1: Address structure in different blockchains (Bitcoin, Ethereum and Diem).

to the blockchain with computing power by expending a certain amount of Central Processing Unit (CPU) cycles to solve a mathematical puzzle. This puzzle is intentionally computationally difficult to solve, yet it is very easy to verify the answer [29].

To add a block of new transactions to the blockchain, a miner must solve the puzzle. The first miner to solve the puzzle sends (proposes) the block to the rest of the network for agreement. If the network agrees on the solution to the puzzle, the miner is rewarded for creating the block and the block is added to the blockchain (the miner wins this round of competition). Through a combination of game theory and economics (effectively betting CPU cycles, which cost money, to win the reward), Proof of Work (PoW) incentivizes consensus instead of attempting to enforce it. Essentially a miner is rewarded for securing the network.

While public blockchains rely on PoW, enterprise (or permissioned) blockchains [15] tend to use the BFT consensus protocols [6]. BFT consensus is based on the idea that a pre-selected, authorized group of validators will create, verify the new blocks.

In a proof of authority consensus model, known participants leverage cryptographical digital signatures to agree upon a set of transactions and their output to advance the blockchain's state [28]. For the Diem Blockchain the set of potential entities that can participate in consensus are known as Validator Owners, while the active participants are known as the Validator Set. The adding and removing of Validator Owners and specifying the current Validator Set is left to the sole discretion of the entity managing "Diem Root" account. Validators receive transactions from clients and share them with each other through a shared mempool protocol.

2.3. Diem Accounting System

In the Diem DLT, an account represents a resource on the Blockchain that can send transactions. Each account is identified by a 16-byte hash value and there are two kinds of accounts, ParentVASP and ChildVASP accounts. The ParentVASP represents the primary account of a digital wallet, while the ChildVASP is defined as the child account of a particular ParentVASP. Multiple ChildVASPs can be created by ParentVASP accounts [20]. Figure 1 represents the address structure in different blockchains.

In Diem, a PoA will be requested from the ParentVASP, and these proofs should include all of their children's assets as well. Table 2 shows the users roles and permission supported by the Diem DLT [13].

Table 2
Diem Roles and Permissions

Role	Granted by	Unique?	Address	Has bal-ances?	Ac-count lim-its?	Fr.able?	Tx pri.
Diem Root	genesis	Globally	0xA550C18	N	-	N	3
Treasury Compliance	genesis	Globally	0xB1E55ED	N	-	N	2
Validator	Diem Root	Per Association member	-	N	-	Y	1
Validator Operator	Diem Root	At most one per Validator	-	N	-	Y	1
Designated Dealer	Treasury Compliance	N	-	Y	N	Y	1
Parent VASP	Treasury Compliance	Per VASP	-	Y	Y	Y	0
Child VASP	Parent VASP	N	-	Y	Y	Y	0

Diem uses a variant of role-based access control (RBAC) to restrict access to sensitive on-chain operations. A role is an entity with some authority in the Diem Payment Network (DPN). Every account in the DPN is created with a single, immutable role that is granted at the time the account is created. Creating an account with a particular role is a privileged operation (e.g., only an account with the ParentVASP role can create an account with the ChildVASP role). In some cases, the role is globally unique (e.g., there is only one account with the Diem Root role). In other cases, there may be many accounts with the given role (e.g., ChildVASP).

2.4. Stablecoin

Stablecoins are cryptocurrencies with the ability to maintain a stable price relative to a particular fiat currency via a “peg mechanism”. A “peg” is a specified price for the rate of exchange between two assets. In the context of currencies, a peg allows foreign currencies to be traded for the chosen base currency at a fixed exchange rate. In the context of cryptocurrency, a peg refers to the specific price that a token is aiming to stay at [9].

Today, stablecoins are mostly used for trading, lending and borrowing crypto assets. They are a crucial component of the decentralized finance (DeFi) – financial services performed by applications on a permissionless blockchain [21].

Stable coins first became widely known as a potential means of global retail payments when Meta (then Facebook) announced its Libra project in 2019. Bitcoin and Ethereum rise and fall by the day and even hour, in contrast, stable coins promise to maintain their value because they are pegged to less volatile assets, like the U.S. dollar or Euro. Because of their potential use as actual currency, U.S. government officials fear the potential risks stable coins pose for consumers and financial markets if they remain unregulated. As an example, the value of the TerraUSD stablecoin (UST) crashed in the cryptocurrency market almost completely at one point on 9 May 2022 and lost its 1 USD peg to the dollar, tanking to a low of 0.02 USD [9] without giving any legal protection to their investors [22].

Stablecoins can be split into three groups according to their collateral and price stabilization mechanisms:

1. off-chain collateralized (e.g. Diem)

2. on-chain collateralized (e.g. Dai)
3. uncollateralized, purely algorithmic stablecoins (e.g. Ampleforth).

The Diem DLT was planned to be an off-chain collateralized project, i.e. it should have used traditional reserve assets to stabilize Libra value, the Diem cryptocurrency. The Diem reserve assets should have been fiat-currency bank deposits and short-term debt, with the US dollar being the most prominent reference currency. As the reserves are not on the blockchain, a custodian is required. In order to maintain price stability, all outstanding stablecoins must be backed by reserve assets. Currently off-chain collateralized stablecoins are Tether, Binance USD and USD Coin.

Unlike the Diem DLT, on-chain collateralized projects back their stablecoins with other crypto assets. They are typically issued by DeFi applications as collateralized debt positions, i.e. a user locks in collateral and in return receives coins created by the application. Thus, the collateral is held directly in the application on the blockchain and no external custodian is needed. Currently an collateralized stable coin system is Dai [19].

Finally, uncollateralized stable coin systems try to keep prices constant by algorithmically adjusting the outstanding number of tokens according to demand. If prices are above the peg, the algorithm will distribute new coins to users, thereby eventually reducing the price. If prices fall below the peg, the system will sell a sort of bond to users in exchange for stable coins. The stable coins received will then be destroyed, leading to a price increase. If prices then move above the peg again, bondholders will be prioritized in the distribution of new coins. In theory, this system incentivizes users to buy bonds if prices fall below the peg and rewards them afterwards as prices exceed the peg again. Currently an uncollateralized stable coin system is Ampleforth [25].

Table 3 shows the blockchains that support stable coins grouped by the stabilization mechanism.

Table 3
Stable coin blockchain grouped by the stabilization mechanism

Blockchain	Crypto Coin	Market Cap (USD)	Stabilization Mechanisms	Max	Min
Diem	Libra	-	Off-Chain Collateralized	-	-
Tether	USDT		Off-Chain Collateralized	1.002	0.999
Binance	BUSD	17,706,848,087.24	Off-Chain Collateralized	1.002	0.998
Dai	DAI	8,855,233,197.17	On-Chain Collateralized	1.010	0.985
Ampleforth	AMPL	87,155,777.68	Uncollateralized	2.11	0.91

3. Research Methodology

The main aim of the study was to better understand the Diem blockchain performance, in terms of number of transactions per second and waiting times. The study presupposes that a blockchain has a better performance than another when the former has a higher number of transactions per second and shorter waiting times when compared to the latter.

Thus, the study was designed to address the following research questions (RQ):

- RQ1: Can the Diem blockchain have a better performance in terms of number of transactions per second when compared to other blockchains, such as Bitcoin and Ethereum?
- RQ2: What are the waiting times to confirm the transactions in the Diem blockchain?

To answer the research questions, the methodology of the study consists of three research phases: a) Data Collection, b) Data Modelling, and c) Data Analysis and Results. The following subsections describe each research phase.

3.1. Data Collection

The Diem DLT provides an application program interface (API) to interact with the blockchain. We developed a script that performs a POST request to the API endpoint (<https://testnet.diem.com/v1>), to collect the data from the Diem blockchain. The script queries the Diem API at regular intervals of 100 milliseconds and it downloads 1000 transaction payloads for each request. A timeout of 100 milliseconds is required to avoid sending too many requests in a given interval of time and receiving the “Too Many Requests” server error. Within the rate-limit of 100 milliseconds, we collected 3.500.000 transactions, which were submitted by Diem users and available on the Diem test network. The same data can also be downloaded from a block explorer, but the collection takes much more time because each request can download just the data of a single transaction. The two block explorers available to download the data transactions are the “InDiem Blockchain Explorer” (<https://indiem.info/explorer>) and the “Diem Blockchain Explorer” (<https://diemexplorer.com/testnet>).

Table 4 shows the summary of the transactions data-set.

The mean, the median, minimum (min), the 25th, 50th, and 75th percentiles and maximum (max) are calculated for each variable shown in the table. Some of these data are the size of the script used to execute the transaction computed in bytes, the gas units used to execute the transaction (gas_used), and the number of transactions added to the Diem blockchain in a one-second interval of time (TPS) [32].

The data were collected as distinct files in JSON format. Listing 1 shows an example of JSON-RPC request used to query the Diem block data. For instance, the second value of the “params” list is an integer value (max=1000) that can be used to limit the number of transactions returned. Listing 2 shows an example of JSON-RPC response used to store the Diem transaction data. The request 1 returns the transactions’ information about a confirmed block in the Diem DLT.

Table 5 describes the structure and the elements of the data related to the user transaction on the Diem test network.

Listing 1 shows an example of JSON-RPC request used to query the Diem block data. For instance, the second value of the “params” list is an integer value that can be used to limit the number of transactions returned; the max value is 1000.

Listing 1: JSON-RPC request to query the Diem DLT

```
{
  // Request: fetches 10 transactions
  curl -X POST -H "Content-Type: application/json" \
  --data '{"jsonrpc": "2.0", "method": "get_transactions", "params": [100000, 10, false], "id": 1}' \
  https://testnet.diem.com/v1
}
```


Table 4

Summaries of the transactions data-set

	file size (B)	gas_used	bytesLength	scriptBytesLength	TPS
Min.	370.0	74.00	834	526.0	0.00
1st Qu.	606.0	74.00	1066	758.0	48.00
Median	606.0	74.00	1066	758.0	60.00
Mean	605.2	77.81	1065	757.2	62.45
3rd Qu.	606.0	74.00	1066	758.0	80.00
Max.	606.0	1748.00	1066	758.0	180.00

Listing 2: Transaction JSON Response

```

{
  "bytes": "00f942c6ed8cab022562617cb36...",
  "gas_used": 479,
  "hash": "af6611b875d2f291c575ff36d...",
  "transaction": {
    "chain_id": 3,
    "expiration_timestamp_secs": 1645710527,
    "gas_unit_price": 0,
    "max_gas_amount": 1000000,
    "public_key": "ae4ccb911d6d36248ee3aedd437...",
    "script": {
      "amount": 1,
      "arguments": [
        "{ADDRESS:_34FCA44C571B29CC0AFF63363609B325}"
      ],
      ...
    },
    "code": "a11ceb0b0...",
    "currency": "XUS",
    "metadata": "",
    "metadata_signature": "",
    "receiver": "34fca44c571b29cc0a...",
    "type": "peer_to_peer_with_metadata"
  },
  "script_bytes": "e001a11ceb0b01...",
  "script_hash": "04ea43107fafc12adcd09...",
  "secondary_public_keys": [],
  "secondary_signature_schemes": [],
  "secondary_signatures": [],
  "secondary_signers": [],
  "sender": "f942c6ed8cab022562617cb361a1ad84",
  "sequence_number": 375,
  "signature": "3e72d6ffc1af77...",
  "signature_scheme": "Scheme::Ed25519",
  "type": "user"
},
"version": 1165000,
"vm_status": { "type": "executed" }
}

```

3.2. Data Modelling

The collected data were not suitable to perform data analysis, because reading these files takes too much time. We organized the data into three .CSV files based on their transaction type. Indeed, in the Diem DLT there are three types of transactions that can be sent by different types of accounts:

- Transactions that send payments to other accounts.
- Transactions that are sent to create accounts, mint and burn Diem Coins.

Table 5
Transactions Properties

Name	Type	Description
sender	string	Hex-encoded account address of the sender
signature_scheme	string	Signature scheme used by the sender to sign the transaction
signature	string	Hex-encoded signature of the transaction signed by the sender
public_key	string	Hex-encoded public key of the transaction sender
secondary_signers	List	Hex-encoded account addresses of the secondary signers
secondary_signature_schemes	List	Signature schemes used by the secondary signers to sign this transaction
secondary_signatures	List	Hex-encoded signatures of this transaction signed by the primary signers
secondary_public_keys	List	Hex-encoded public keys of the secondary signers
sequence_number	unsigned int64	Sequence number of this transaction corresponding to sender's account
chain_id	unsigned int8	Chain ID of the Diem network. The chain ID is a property of the chain managed by the node. It is used for replay protection of transactions.
max_gas_amount	unsigned int64	Maximum amount of gas that can be spent for the transaction
gas_unit_price	unsigned int64	Maximum gas price to be paid per unit of gas
gas_currency	string	Gas price currency code
expiration_timestamp_secs	unsigned int64	The expiration time (Unix Epoch in seconds) for the transaction
script_hash	string	Hex-encoded sha3 256 hash of the script binary code bytes used in the transaction
script_bytes	string	Hex-encoded string of BCS bytes of the script. BCS (formerly "Libra Canonical Serialization" or LCS) is a serialization format developed in the context of the Diem blockchain.
script	Script	The transaction script and arguments of this transaction

- Transactions that help account recovery, key rotation, by adding currencies and other account administration tasks.

An account can send a payment to another account by submitting a transaction. If an account A wishes to send a payment to another account B, it can do so by executing a "peer_to_peer_with_metadata" transaction script. If an account A (the ParentVASP account) wishes to create another account B (a ChildVASP account), it can do so by executing a "create_child_vasp_account" transaction script with a single ParentVASP account, a user can create up to 256 ChildVASP accounts. The transaction script allows you to specify: Which currency the new account should hold, or if it should hold all known currencies. If the user wants to initialize the ChildVASP account with a specified amount of coins in a given currency. An individual can have at most one root account per Regulated VASP. Diem Networks was suppose to create a ParentVASP account via the personal authentication key abd via the "create_parent_vasp_account" transaction script. Table 6 shows the number of transactions type found in the collected dataset.

Table 6
Types of Transactions

	transaction type	occurencies
1	create_child_vasp_account	112 320
2	create_parent_vasp_account	1 230
3	peer_to_peer_with_metadata	1 180 980

3.3. Analysis and Results

The section presents the analysis of the transactions data as modelled in the previous section. The data sets are stored in a tabular format where the rows (around one million) represent the different transactions and the columns (nine) represent their characteristics. The total size of the database is 58,1 Mega-Byte and is publicly available via Zenodo [30].

For blockchain-based applications, scalability has been extensively studied since the introduction of Bitcoin [8, 42]. We scraped and analysed the data from the Diem DLT API to compute the scalability of the Diem blockchain. Unlike other blockchains, the Diem blockchain can operate in either “normal” or “recovery” mode [2]. When the Diem DLT is on “normal mode”, blocks with transactions are generated and committed in sequence. The system can switch to recovery mode in case of a failing validator node or when the system is under attack. During this time, the performance of the Diem blockchain can be negatively impacted or the processing of transactions can be temporarily put to a stop.

A previous study [2] developed a simulation model to estimate how close Diem is to realizing its goals. They calculated the amount of time a user has to wait to receive confirmation that a transaction made on the blockchain will not be changed. The results showed that, for 100 validators, that amount of time is 10 seconds. As it comes to transaction throughput, the Diem blockchain still requires major improvements, as in the best case only 300 transactions per second were estimated for 100 validators.

Another study [40] have set up an infrastructure made of physical servers (14 cores with 384GB of RAM) to measure the number of transactions Libra DLT can process in a particular time span. They have shown that the Libra blockchain can process about one thousand transactions per second at most (one validator active), but the performance drops significantly as the number of validators increases (350 TPS with 16 validators). They compared their results with other permissioned blockchains and they found in particular that Diem has worse performance when compared to the Hyperledger Fabric.

Table 7 below shows the TPS and average transaction confirmation time of Diem DTL vs. other blockchains. The data about the Bitcoin and Ethereum blockchains have been taken from different academic works.

As depicted in table 7, Ethereum has a transaction speed of 15.6 transactions per second.

The rate at which valid transactions are confirmed per second in the Ethereum blockchain is higher when compared to Bitcoin. However, the TPS of Ethereum is low compared to the TPS of Diem DLT, which has over 60 transactions executed per second. Figure 2a shows the number of transactions that the Diem test network can process each second (TPS).

Figure 2c shows the power complementary cumulative distribution (CCDF) as a function of the transaction waiting times in the memory pool before being confirmed in the Diem Blockchain.

Table 7
Diem TPS Comparison against Bitcoin, Ethereum

	mean (TPS)	max	min	std
Bitcoin	4.60	-	-	-
Ethereum	15.60	-	-	-
Diem (test network)	65.51	185	0	35.72
Diem (simulation model)	80.00	300	0	-
Diem (14 cores, 384GB RAM, 16 peers)	350	-	-	-

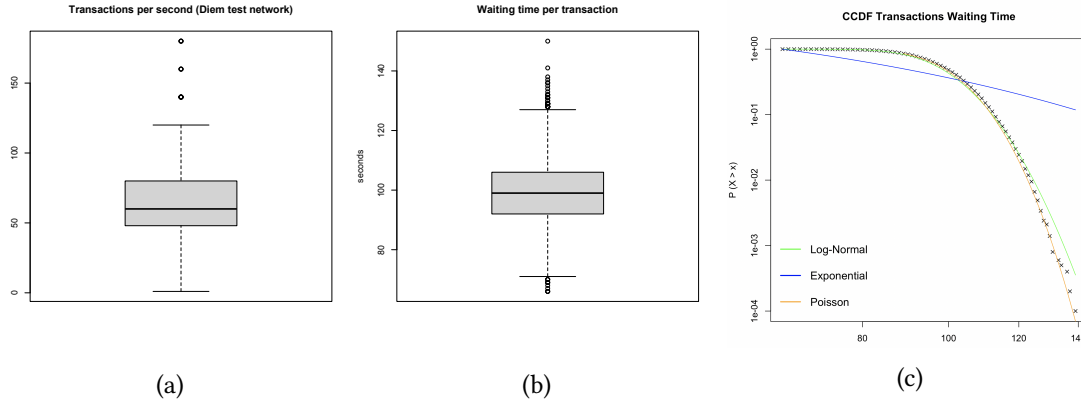


Figure 2: (a) Transactions per second in the Diem test network. (b) Waiting time per transaction. (c) CCCDF of the Waiting time per transaction

The empirical CCDF seems to be well fitted by Poisson’s law shown as the continuous thin orange line curving downward. Other academic studies on other blockchain suggest that the waiting time for transactions in the memory pool has a trend that follows Poisson’s law [31].

4. Conclusion

Blockchain technology is rapidly evolving. Understanding the core components of the technology and how they work together is crucial to make it available also to a larger audience [43]. Each component of the blockchain system plays an important role in the technology stack. This study sheds light on some components of the Diem DLT, such as the consensus and the specificity of the Move programming language used to write smart contracts.

According to some academic sources [36, 34], the project failed for political-economical reasons. Nonetheless, some ideas of the project have been adopted and could be adopted by other blockchains. For instance, the Diem consensus allows having a better TPS when compared to other blockchains, such as Ethereum and Bitcoin. Moreover, unlike the consensus mechanism adopted by other blockchains, such as Ethereum and Bitcoin, the Diem BFT consensus protocol allows being compliant with the law in order to achieve large-scale adoption.

The data collection and analysis of the Diem transactions, even though performed on the

test network, show that the transaction throughput, expressed as a number of transactions per second, is better when compared to other popular blockchains. This is very important to achieve large-scale adoption of this technology, as it can support a larger number of transactions. Another important characteristic of Diem blockchain, that has already been taken as a model by other blockchains, is the use of traditional reserve assets, such as government bonds, and stable fiat currencies, like the USD, to make the cryptocurrency value stable.

Finally, the programming language Move allows for defining custom resource types. This feature helps smart contract developers write business logic for wrapping assets and enforce access control policies without using external libraries. For all these reasons, the study can provide useful insights for any blockchain developers to choose the right components for a successful blockchain adoption at a larger scale.

Acknowledgments

This research was supported by “Fondazione di Sardegna” through the project “Analysis of innovative Blockchain technologies: Libra, Bitcoin and Ethereum” (CUP: F72F20000190007).

References

- [1] Marco Baldi, Franco Chiaraluce, Emanuele Frontoni, Giuseppe Gottardi, Daniele Sciarroni, and Luca Spalazzi. Certificate validation through public ledgers and blockchains. In *ITASEC*, pages 156–165, 2017.
- [2] Jeanpierre Balster. Investigating the scalability of the diem blockchain: A simulation approach. *Eindhoven University of Technology Press*, 2021.
- [3] Massimo Bartoletti, Letterio Galletta, and Maurizio Murgia. A true concurrent model of smart contracts executions. In *International Conference on Coordination Languages and Models*, pages 243–260. Springer, 2020.
- [4] Sam Blackshear, Evan Cheng, David L Dill, Victor Gao, Ben Maurer, Todd Nowacki, Alistair Pott, Shaz Qadeer, Dario Russi Rain, Stephane Sezer, et al. Move: A language with programmable resources. *Libra Assoc.*, 2019.
- [5] Dirk Bullmann, Jonas Klemm, and Andrea Pinna. In search for stability in crypto-assets: are stablecoins the solution? *ECB Occasional Paper*, 1(230), 2019.
- [6] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.
- [7] Panagiotis Chatzigiannis and Konstantinos Chalkias. Proof of assets in the diem blockchain. In *International Conference on Applied Cryptography and Network Security*, pages 27–41. Springer, 2021.
- [8] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and scalability. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 122–128. IEEE, 2018.
- [9] Ryan Clements. Built to fail: The inherent fragility of algorithmic stablecoins. *Wake Forest L. Rev. Online*, 11:131, 2021.

- [10] Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Engineering trustable choreography-based systems using blockchain. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1470–1479, 2020.
- [11] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain. *University of Southampton Institutional Repository*, 2018.
- [12] Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. Smart contracts vulnerabilities: a call for blockchain software engineering? In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 19–25. IEEE, 2018.
- [13] LLM Eleni Katopodi et al. Blockchain market: Regulatory concerns arising from the ‘diem’ example in the field of free competition 1. *EU and Comparative Law Issues and Challenges Series*, pages 197–216, 2021.
- [14] Brandon Williams et al. Aptos. <https://github.com/aptos-labs/aptos-core>, 2022.
- [15] Claudio Ferretti, Alberto Leporati, Luca Mariot, and Luca Nizzardo. Transferable anonymous payments via tumblebit in permissioned blockchains. In *DLT@ITASEC*, pages 56–67, 2019.
- [16] Stefano Ferretti and Gabriele D’Angelo. On the ethereum blockchain structure: A complex networks theory perspective. *Concurrency and Computation: Practice and Experience*, 32(12):e5493, 2020.
- [17] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Blockchain-based database to ensure data integrity in cloud computing environments. In *ITASEC*, pages 146–155, 2017.
- [18] Florian Idelberger, Guido Governatori, Régis Riveret, and Giovanni Sartor. Evaluation of logic-based smart contracts for blockchain systems. In *International symposium on rules and rule markup languages for the semantic web*, pages 167–183. Springer, 2016.
- [19] Clemens Jeger, Bruno Rodrigues, Eder Scheid, and Burkhard Stiller. Analysis of stablecoins during the global covid-19 pandemic. In *2020 Second International Conference on Blockchain Computing and Applications (BCCA)*, pages 30–37. IEEE, 2020.
- [20] Kim Peiter Jørgensen and Roman Beck. Universal wallets. *Business & Information Systems Engineering*, pages 1–11, 2022.
- [21] Ayten Kahya, Bhaskar Krishnamachari, and Seokgu Yun. Reducing the volatility of cryptocurrencies—a survey of stablecoins. *arXiv preprint arXiv:2103.01340*, 2021.
- [22] Evan Kereiakes, Marco Di Maggio Do Kwon, and Nicholas Platias. Terra money: Stability and adoption, 2019.
- [23] Shafaq Naheed Khan, Faiza Loukil, Chirine Ghedira-Guegan, Elhadj Benkhelifa, and Anoud Bani-Hani. Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-peer Networking and Applications*, 14(5):2901–2925, 2021.
- [24] Jin-Whan Kim. Analysis of blockchain ecosystem and suggestions for improvement. *Journal of information and communication convergence engineering*, 19(1):8–15, 2021.
- [25] Evan Kuo, Brandon Iles, and Manny Rincon Cruz. Ampleforth: A new synthetic commodity. *Ampleforth White Paper*, 2019.
- [26] Andrea Lisi, Andrea De Salve, Paolo Mori, and Laura Ricci. A smart contract based

- recommender system. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 29–42. Springer, 2019.
- [27] Damiano Di Francesco Maesa and Paolo Mori. Blockchain 3.0 applications survey. *Journal of Parallel and Distributed Computing*, 138:99–114, 2020.
- [28] Lara Mauri, Stelvio Cimato, and Ernesto Damiani. A comparative analysis of current cryptocurrencies. In *ICISSP*, pages 127–138, 2018.
- [29] Satoshi Nakamoto. Bitcoin v0. 1 released. *The Mail Archive*, 9, 2009.
- [30] Pierro. Diem blockchain transactions data set, June 2022.
- [31] Giuseppe Antonio Pierro, Henrique Rocha, Stéphane Ducasse, Michele Marchesi, and Roberto Tonelli. A user-oriented model for oracles’ gas price prediction. *Future Generation Computer Systems*, 128:142–157, 2022.
- [32] Giuseppe Antonio Pierro, Henrique Rocha, Roberto Tonelli, and Stéphane Ducasse. Are the gas prices oracle reliable? a case study using the ethgasstation. In *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 1–8. IEEE, 2020.
- [33] Giuseppe Antonio Pierro, Roberto Tonelli, and Michele Marchesi. An organized repository of ethereum smart contracts’ source codes and metrics. *Future internet*, 12(11):197, 2020.
- [34] Marc Pilkington. From libra 1.0 to libra 2.0 (diem): between programmed failure and renewed relevance for political economy. *Revue d’Economie Politique (forthcoming)*, 2022.
- [35] Ivan Pupo. From libra to diem. the pursuit of a global private currency. *Global Jurist*, 2021.
- [36] Yubin Qu, W Eric Wong, and Dongcheng Li. Empirical research for self-admitted technical debt detection in blockchain software projects. *International Journal of Performability Engineering*, 18(3), 2022.
- [37] Jahja Rrustemi and Nils S Tuchschnid. Facebook’s digital currency venture “diem”: the new frontier... or a galaxy far, far away? *Technology innovation management review*, 10(12), 2020.
- [38] Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Publicly verifiable proofs from blockchains. In *IACR International Workshop on Public Key Cryptography*, pages 374–401. Springer, 2019.
- [39] Roberto Tonelli, Giuseppe Destefanis, Michele Marchesi, and Marco Ortu. Smart contracts software metrics: a first study. *arXiv preprint arXiv:1802.01517*, 2018.
- [40] Jiashuo Zhang, Jianbo Gao, Zhenhao Wu, Wentian Yan, Qize Wo, Qingshan Li, and Zhong Chen. Performance analysis of the libra blockchain: An experimental study. In *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*, pages 77–83. IEEE, 2019.
- [41] Weijie Zhao. Blockchain technology: development and prospects. *National Science Review*, 6(2):369–373, 2019.
- [42] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *Ieee Access*, 8:16440–16455, 2020.
- [43] Mirko Zichichi, Stefano Ferretti, and Gabriele D’Angelo. A distributed ledger based infrastructure for smart transportation system and social good. In *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2020.

DELTA - Distributed Elastic Log Text Analyser

Piergiuseppe Di Pilla¹, Remo Pareschi^{1,2}, Francesco Salzano¹ and Federico Zappone^{1,2}

¹*Stake Lab, University of Molise, Campobasso, Italy*

²*BB-Smile Srl, Rome, Italy*

Abstract

Distributed systems have become ubiquitous in recent years, with those based on distributed ledger technology (DLT), such as blockchains, gaining more and more weight. Indeed, DLT ensures strong data integrity thanks to complex cryptographic protocols and high distribution. That said, even the most powerful systems will never be perfect, and, in fact, the larger they get, the more exposed they become to threats. For traditional systems, log auditing effectively addresses the problem and makes it possible to analyze the use of applications. However, DLT systems still lack a wide range of log analyzers due to the particularities of their distribution. To help remedy this weakness, we propose here a generic auditing system called *DELTA* (for Distributed Elastic Log Text Analyzer). By coupling Natural Language Processing with the *Docker Engine* of the *Filebeat*, *Logstash stack*, *Elasticsearch* and the visual tool *Kibana*, *DELTA* tracks, analyzes and classifies logs generated by DLT systems. Additionally, it enables real-time monitoring thanks to visual analysis and querying of structured data. *DELTA* is the first auditing system applicable to blockchains that can be integrated with the *Docker Engine*. In addition to describing its general principles and specific components, we illustrate its application to Hyperledger Fabric, the most popular of the platforms for building private blockchains.

Keywords

Distributed Ledger Technology, Log Analysis, Cybersecurity, Natural Language Processing, Blockchain, NLP, DLT .

1. Introduction

Distributed systems have been spreading rapidly in recent years [30], and the emergence of Distributed Ledger Technologies (DLTs) such as blockchains have strongly contributed to this trend. These technologies find a wide range of possible applications in areas such as the Internet of Things (IoT), healthcare, supply chain management, energy, genomics, fintech, insurance, automotive, etc. [2, 27, 21, 8, 5, 15, 13]. As a consequence, there is an ongoing strengthening of development frameworks such as *Ethereum*, *Hyperledger*, *EOSIO*, *Corda*, *Waves*, *Quorum* etc. which are constantly adding new features.

The trend is explained by the ability of DLT to provide a high degree of security, compared to classical systems, by encrypting and decentralizing data, aspects that are both paramount

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

† These authors contributed equally.

✉ p.dipilla@studenti.unimol.it (P. D. Pilla); remo.pareschi@unimol.it (R. Pareschi); f.salzano1@studenti.unimol.it (F. Salzano); federico.zappone@unimol.it (F. Zappone)

🌐 <http://docenti.unimol.it/index.php?u=remo.pareschi> (R. Pareschi); <https://github.com/ZappaBoy> (F. Zappone)

🆔 0000-0001-6455-6575 (F. Zappone)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

for the development of decentralized applications (*dApps*). Data security is a crucial issue for information systems in general, so, over time, numerous tools have been developed for data protection and monitoring system access, including auditing, which provides a wealth of security-related information. In particular, system log auditing extracts information about the operations carried out and the conditions in which they took place and keeps track of their timelines. Logs are therefore essential for analyzing the behavior of IT systems under both normal and abnormal conditions. Indeed, while the normal case provides a history of the operations carried out, the anomalous one helps identify system errors and detect vulnerabilities, thus preventing cyberattacks.

However, compared to development frameworks, distributed systems auditing tools lag, especially when it comes to blockchains. The more established ecosystems, such as Bitcoin and Ethereum, have their log analysis tools. Yet, there is a lack of a standardized tool that can be integrated with most frameworks and is endowed with real-time monitoring capabilities. Existing tools designed for cloud and decentralized systems [18] are not easy to integrate with development frameworks for blockchain applications and are not up to the challenges regarding complete log auditing of blockchain systems [7]. We started from these premises in carrying out the design and development of a universal log analysis tool, which takes the name of *DELTA* for Distributed Elastic Log Text Analyzer, aimed at analyzing logs of activities on most of the existing development frameworks for distributed and non-distributed systems - a versatility which is made possible thanks to the use of the *Docker Engine* and the *stack ELK* (*Elasticsearch*, *Logstash Kibana*) integrated via *Filebeat*.

For this purpose, we make use of the *Docker Engine* as a bridge for collecting logs between the analyzer and distributed systems. Thanks to *Docker*, it is, in fact, possible not only to integrate completely different systems but also to analyze the logs produced through the *ELK* stack. This stack makes it possible to efficiently control log collection methods by accessing *Docker containers*. Furthermore, *Filebeat* takes care of managing log collection methods in real-time, and *Logstash* enables automatic log insertion into the *Elasticsearch* database, which in turn supplies data to *Kibana* for immediate viewing through a customizable graphical interface.

The developed tool is not limited to traceability in that the textual part of the traced logs is subjected to analysis through Natural Language Processing (NLP). NLP is used to perform, upon the text within logs produced by the *Docker containers*, three types of analysis, namely: keyword extraction, classification, and sentiment analysis. Keywords are extracted through two different models: the more precise *KeyBERT*, based on *BERT* (Bidirectional Encoder Representations from Transformers), and the more versatile *YAKE!* [6]. As regards log classification, the choice fell on the *Zero-Shot facebook / bart-large-mnli* developed by Meta (formerly Facebook), which works without requiring data outside the text. The idea of *Zero-Shot* models is to analyze and classify data that are also completely different from those with which the training was carried out using the methodology for statistical inference described in [20]. Finally, sentiment analysis is performed through *VADER* (Valence Aware Dictionary and sEntiment Reasoner), an open-source analysis tool based on rules for extracting sentiment using dictionaries. All log collection, analysis, and classification processes occur in real-time, enabling interaction with resource monitoring processes. In this way, it is possible to focus log analysis on the security problems and undertake mitigating actions as needed.

Structure of the paper. The remaining part of the article is organized as follows: Section 2

describes the methodology underlying *Delta* and the components used for its implementation; Section 3 describes the application of DELTA to Hyperledger Fabric, the most popular platform for private blockchains; Section 4 describes future work; Section 5 concludes the paper.

2. Methodology

The developed system is mainly divided into two macro sections, the first relating to the collection of logs through the use of the *Elastic* components and the second part consisting of the textual analysis systems provided by the *DELTA* tool. The following sections describe the methodologies used to implement the system: Section 2.1 describes the use of the *Elastic* stack and the log flow within the system, Section 2.2 is instead dedicated to the illustration of the developed tool and how the log analysis and their re-elaboration takes place.

2.1. Elastic stack

The acquisition of logs produced by distributed services and systems was managed through the combined use of several *Elastic* components. In fact, the stack used includes 4 such components, namely *Filebeat*, *Logstash*, *Elasticsearch* and *Kibana*. The *Filebeat* component has been added on top of the standard *ELK* stack as it supports log extraction from highly heterogeneous contexts and therefore is perfectly suited to distributed environments.

These components fit together to obtain, manage, index, and view the generated logs automatically and instantly. The logs come from the containers where the distributed services are located and are first extracted through *Filebeat*, which reads the data directly by connecting to the log files managed by the *Docker engine*. Then *Filebeat* passes the logs to *Logstash* which aggregates them and places the raw information inside *Elasticsearch* creating a special index for archiving. Finally, *Elasticsearch* makes indexing of the entered data available for access via *Kibana* which provides the visual analysis of both the aggregate data entered through the collection process and of the analyzed data (Figure 1).

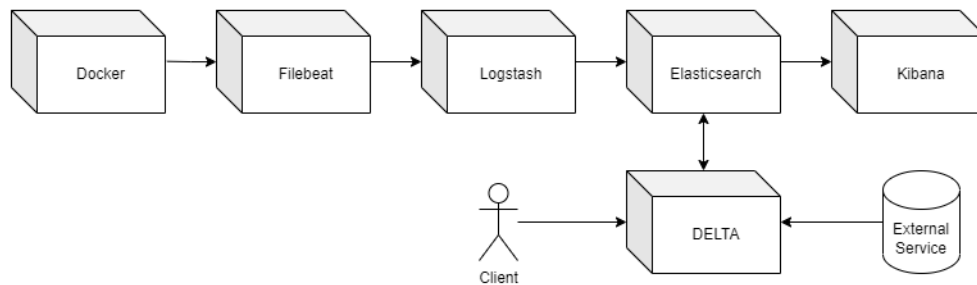


Figure 1: Structure of the system

2.2. DELTA Analyser

DELTA is aimed at processing the logs in the Elasticsearch database to extract relevant information using Natural Language Processing techniques to facilitate data monitoring and analysis operations. This occurs through a continuous activity of detection of relevant patterns such as the presence of IP addresses, as well as of processing on the following three dimensions (Figure 2):

- **Keywords Extraction**
- **Category Classification**
- **Sentiment Analysis**

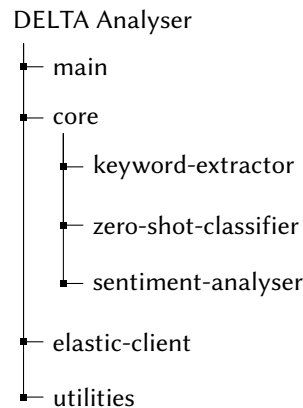


Figure 2: Structure of DELTA tool

2.3. Keywords Extraction

Keyword extraction picks out words with the highest informative impact on the text, thus making it possible to conduct statistical analysis or associate keywords with the triggering of events across vastly varying contexts. Moreover, keywords can be exploited to boost monitoring effectiveness to access the logs by filtering their content.

Many methods, techniques, and algorithms extract keywords or key phrases from texts (e.g., TF-IDF, Rake, YAKE!). Since *DELTA* was designed for a generic context, we cannot make predictions about the input to be analyzed. Vice versa, given the specificity that characterizes some application contexts, there may be extraction models based on statistical concepts that ignore specific keywords. When dealing with a distributed system that uses specific terms to identify its components (for example, the word *Tangle* in the context of the distributed ledger *IOTA*), such elements would be ignored by the most models based on word frequency or dictionaries. These considerations led to implementing both a keyword extraction technique based on the semantic similarity of the words and a method based on the statistical properties of the texts. Therefore, for keyword extraction, we relied on two models, namely *KeyBERT* and *YAKE!*.

2.3.1. KeyBERT

KeyBERT is based on *BERT* (Bidirectional Encoder Representations from Transformers) [17], published by Google in 2018, a model that uses a *transformer* [28] architecture to map words, phrases and text into number vectors that capture their meaning. *BERT* is a bidirectional model, which makes it able to interpret a word based on the context of the sentence, regardless of whether the relevant information is left or right, unlike left-to-right architectures, which look only at the words preceding the one being processed [12]. Furthermore, *BERT* does not resort to recursive levels unlike *LSTM*-based technologies [16], but instead exploits the *Self-Attention* [9] mechanism. The *KeyBERT* system can be implemented with different *transformer* models, with the basic model *all-MiniLM-L6-v2* having limited needs for computational resources with a trade-off in precision levels. While, according to the official documentation, the highest quality model turns out to be *all-mpnet-base-v2*, we have chosen to use the less powerful *paraphrase-albert-large* model, as log texts are generally compact, so this model achieves excellent accuracy with lower resource consumption.

2.3.2. YAKE!

YAKE! or *Yake!* or *Yake*, is an automatic keyword extraction algorithm that stands out above all for its simplicity and an excellent balance between computational resource requirements and quality of the analytics. It is an unsupervised algorithm based on statistical textual characteristics extracted from individual documents. Therefore, it does not need to be trained on a particular set of documents, nor does it depend on external dictionaries and *corpora*, nor has limitations as regards text sizes, languages, or domains. It also does not use *Part of Speech Tagging* [26], which makes it language-independent, except for the use of different but static stopword lists for each language. This makes it easy to apply to languages other than English, particularly low-diffusion languages for which open-source language processing tools may be underperforming.

2.3.3. KeyBERT compared to YAKE!

KeyBERT and *YAKE!* thus provide alternative models for keyword extraction. In testing the two algorithms on logs, we found out that both offer high accuracy for short texts. Nevertheless, the model suggested by default is *KeyBERT*, in virtue of its higher accuracy for longer texts. On the other hand, *KeyBERT* turns out to be significantly more onerous in terms of performance and waiting time. Therefore, *DELTA* provides both approaches to let users choose the most suitable one for the analysis context.

2.4. Log Classification

Classifying logs according to labels (i.e., classification categories) can help analyze and monitor distributed systems. First, it is thus possible to get an idea of the frequency with which logs that share the same label occur to facilitate the identification of related problems. Furthermore, through the analysis of the logs of the same category, it is possible to identify the presence of specific patterns, which can then be used to verify the system's correct functioning or detect anomalies attributable to errors or tampering attempts. Finally, labels offer a way to access

content in addition to keyword-based querying. The classification approach used in *DELTA* is a hybrid that combines machine learning with rules, being a *Zero-Shot* [24] classifier. As illustrated in [29], this methodology classifies text, documents, or sentences without resorting to any previously tagged data by using a natural language processing model, pre-trained in an environment or domain that may be completely different from the application domain. This makes it possible to classify texts from heterogeneous contexts. It provides a probabilistic value of whether the text belongs to a label by taking a text and a list of possible labels as input. Thus, through a threshold, the text is labeled according to the categories to which it belongs. The way *Zero-Shot* is used in *Delta* provides for multiple labels to be assigned to a single log with different probabilistic scores by exploiting the fact that the model output is an independent probabilistic value for each supplied label. This gives a more detailed view of the system behavior and the consequent possibility of monitoring logs in a more specific and targeted way. There are five preset tags used within *Delta*: *Security*, *Connection*, *Communication*, *Transaction* and finally *Operation*. However, these can be changed according to the context of use. The currently adopted model is the one provided by Facebook: 'facebook/bart-large-mnli' [1], but this too can be changed according to needs and preferences.

2.5. Sentiment Analysis

In addition to keyword extraction and classification analysis, log sentiment analysis is also performed. Sentiment analysis consists of language processing and analysis aimed at identifying the subjectivity value of the text, with the primary goal to determine the polarity of a document, i.e., to classify it according to the degree of positivity, negativity, or neutrality of the concepts expressed. As surprising as it may seem, logs carry sentiment that can usefully shed light on what is going on within the monitored system. In order to detect anomalies and errors, a monitoring system based on *DELTA* will indeed benefit from the identification of logs loaded with negative sentiment, indicative of errors or malfunctions. For the extraction of sentiment from the logs, the library *VADER-lexicon general* is used, which, born as a sentiment analysis library aimed at social media and customer feedback, has valuable features that make it an excellent analyzer for short texts and hence for logs too. Once a given text has been analyzed, *VADER* responds with a polarity value called *compound* that indicates the degree of positivity or negativity of the sentiment of the analyzed text. The *compound* is later processed to define a sentiment evaluation label through thresholds that were tuned and set according to empirical evidence. Within *DELTA*, it is also possible either to modify the values of the thresholds or to add new ones to refine levels of positivity and/or negativity.

2.6. Additional Log Processing

In addition to the textual analysis of the logs, further processing was carried out to bring intrinsically relevant information to the fore. First of all, elements deemed irrelevant for the analysis were removed. We also worked on the *Logstash* component responsible for collecting data and creating a very detailed structure containing all the possible information directly observable from the log generation sources. This structure has been stripped down and simplified as far as possible to facilitate future analyses and speed up information sharing. Furthermore,

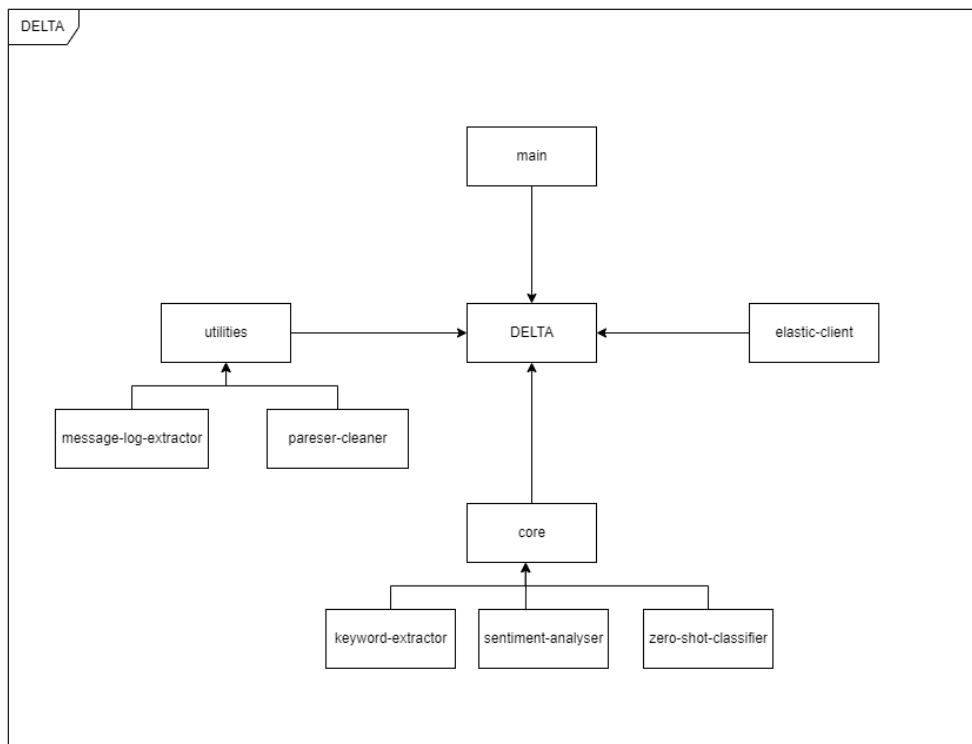


Figure 3: Structure of the system

regarding security aspects and the control of the use of the system, the extraction of IP addresses and connection ports, if present in the logs, was carried out. Finally, the information relating to the log output standard was also extracted, thus making it possible for the aggregation filters to operate in a simplified manner based on the type of log produced by the systems (Listing 1).

3. Application to Hyperledger Fabric

We briefly describe a *DELTA* application to Hyperledger Fabric [4, 11], the most adopted platform for building private distributed ledgers. The challenges presented by blockchain and DL management were, in fact, the initial motivation for *DELTA*, even if its construction was then generalized to all systems that can be containerized and distributed through technologies such as Docker and Kubernetes.

Like any distributed ledger, Fabric’s goals include ensuring a secure environment. However, it has known vulnerabilities [10, 22, 3] that provide attack points for malicious users. While these can be mitigated, there is a lack of a monitoring system that can detect potential attacks and act promptly. This problem can be addressed by using *DELTA* to analyze the logs produced by the system during the attack phase. Log checking is performed to identify attack patterns, and then these are used to develop a monitoring system to spot and mitigate threats in real-time.

Since Fabric blockchains are distributable using the Docker engine, *DELTA* is a close fit for log analysis of the entire Fabric network. The monitoring system is based on the logs generated by the Docker containers and then analyzed and processed by *DELTA* in real-time. Queries can filter logs and obtain only the relevant ones for attack detection. To this aim, several additional valuable data are extractable, such as name, unique ID image, execution status, and installed volumes of Docker containers. Moreover, *DELTA* provides keywords, sentiment, and the types of the log, which can be Security, Connection, Communication, Transaction, and Operation.

The attacks on Fabric fall into two broad categories, depending on whether they are about the network rather than the execution of smart contracts [23]. *DELTA* is particularly effective at dealing with the first ones. They are essentially variants of well-known attacks in distributed systems, e.g., Distributed Denial of Service[19], Sybil[14] and Eclipse attacks [25], which exploit some specificities of Fabric, such as the relatively lower level of decentralization, compared to other blockchains, resulting from design choices like the use of a centralized Ordering Service for transaction management. A monitoring system aimed at network attacks and consisting of three microservices was consequently implemented, namely *i*) a service that relies on *DELTA* to detect patterns of potential danger, once anomalous behavior is confirmed, sends a warning to all configured addresses, *ii*) a service that takes care of sending warning messages via webhook based on the detections made as in *i*), *iii*) a dedicated mitigation service.

4. Future Work

The *DELTA* tool provides an initial log auditing approach specific to distributed systems with a focus on blockchain and *DLT* platforms. However, it is limited to the *Docker engine*. Although widely used and suitable for distributed systems, *Docker* does not scale effectively to very large systems. Consequently, the next step will be to integrate *DELTA* with *Kubernetes*, an open-source platform, initially developed by *Google*, for managing workloads and orchestrating containerized services, which simplifies both system configuration and automation of service delivery practices in very large systems.

5. Conclusion

The exponential growth of distributed systems in recent years, mainly due to the advent of Distributed Ledger Technology and, in particular, blockchains, has led to greater attention to these systems' security and analysis issues. In particular, the security of the information present within distributed systems is paramount because these systems are being increasingly deployed into contexts characterized by sensitive information. For this purpose, we have designed and implemented the *DELTA* tool that collects and stores logs generated by the services that make up the system through the use of some of the components provided by the *Elastic* search ecosystem for data analytics. Then the logs are suitably processed to simplify their analysis. Finally, their text content goes through Natural Language Processing to extract keywords and sentiment and is classified according to relevant categories. Keywords enable effective log search, and their extraction can be done according to needs by choosing between *KeyBERT*, more precise, and *YAKE!*, faster and lighter. Sentiment analysis is performed through the *VADER* algorithm to

measure the degree of text sentiment, where a significant degree of negativity warns about the need to carry out thorough checks on what is happening to the system. Logs are classified in categories that can be set based on the characteristics of the execution system to access them according to classification.

The purpose of these analytical capabilities is to effectively provide the information extracted from the logs to external monitoring processes, which can thus carry out specific and detailed analyses based on the problems at hand and consequently mitigate them. To this end *DELTA* was made customizable and is interfaceable with other platforms through *REST APIs* to query the system and suitably filter content.

References

- [1] Stanislaw Adaszewski, Pascal Kuner, and Ralf J Jaeger. Automatic pharma news categorization. *arXiv preprint arXiv:2201.00688*, 2021.
- [2] Shiroq Al-Megren, Shada Alsalamah, Lina Altoaimy, Hessah Alsalamah, Leili Soltanisehat, Emad Almutairi, et al. Blockchain use cases in digital sectors: A review of the literature. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1417–1424. IEEE, 2018.
- [3] Nitish Andola, Manas Gogoi, S Venkatesan, Shekhar Verma, et al. Vulnerabilities on hyperledger fabric. *Pervasive and Mobile Computing*, 59:101050, 2019.
- [4] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.
- [5] Paolo Bottoni, Nicola Gessa, Gilda Massa, Remo Pareschi, Hesham Selim, and Enrico Arcuri. Intelligent smart contracts for innovative supply chain management. *Frontiers in Blockchain*, 3:52, 2020.
- [6] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289, 2020.
- [7] Michael P Cangemi and Gerard Brennan. Blockchain auditing—accelerating the need for automated audits! *EDPACS*, 59(4):1–11, 2019.
- [8] Federico Carlini, Roberto Carlini, Stefano Dalla Palma, Remo Pareschi, and Federico Zappone. The genesy model for a blockchain-based fair ecosystem of genomic data. *Frontiers in Blockchain*, 3:57, 2020.
- [9] Baiyun Cui, Yingming Li, Ming Chen, and Zhongfei Zhang. Fine-tune BERT with sparse self-attention mechanism. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3548–3553, Hong Kong, China, November 2019. Association for Computational Linguistics.
- [10] Ahaan Dabholkar and Vishal Saraswat. Ripping the fabric: Attacks and mitigations

- on hyperledger fabric. In *International Conference on Applications and Techniques in Information Security*, pages 300–311. Springer, 2019.
- [11] Stefano Dalla Palma, Remo Pareschi, and Federico Zappone. What is your distributed (hyper) ledger? In *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 27–33. IEEE, 2021.
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.
- [13] Ali Dorri, Marco Steger, Salil S Kanhere, and Raja Jurdak. Blockchain: A distributed solution to automotive security and privacy. *IEEE Communications Magazine*, 55(12):119–125, 2017.
- [14] John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002.
- [15] Samuel Fosso Wamba, Jean Robert Kala Kamdjoug, Ransome Epie Bawack, and John G Keogh. Bitcoin, blockchain and fintech: a systematic review and case studies in the supply chain. *Production Planning & Control*, 31(2-3):115–142, 2020.
- [16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.
- [17] Prashant Johri, Sunil K Khatri, Ahmad T Al-Taani, Munish Sabharwal, Shakhzod Suvanov, and Avneesh Kumar. Natural language processing: History, evolution, application, and future work. In *Proceedings of 3rd International Conference on Computing Informatics and Networks*, pages 365–375. Springer, 2021.
- [18] Łukasz Kufel. Tools for distributed systems monitoring. *Foundations of Computing and Decision Sciences*, 41(4):237–260, 2016.
- [19] Felix Lau, Stuart H. Rubin, Michael H. Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: "Cybernetics Evolving to Systems, Humans, Organizations, and their Complex Interactions", Sheraton Music City Hotel, Nashville, Tennessee, USA, 8-11 October 2000*, pages 2275–2280. IEEE, 2000.
- [20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
- [21] Giorgio Alessandro Motta, Bedir Tekinerdogan, and Ioannis N Athanasiadis. Blockchain applications in the agri-food domain: the first wave. *Frontiers in Blockchain*, 3:6, 2020.
- [22] Cathrine Paulsen. Revisiting smart contract vulnerabilities in hyperledger fabric. <https://cse3000-research-project.github.io/static/d23cb4583e6d97a1e509eafda859c424/poster.pdf>, 2021.
- [23] Pierluigi Di Pilla, Remo Pareschi, Francesco Salzano, and Federico Zappone. Hyperledger fabric attacks mitigation (extended abstract). In *FOCODILE 2022 - 3rd International Workshop on Foundations of Consensus and Distributed Ledgers*, 2022.
- [24] Amit Chaudhary published in amitnes. Zero shot learning for text classification.
- [25] Atul Singh, Miguel Castro, Peter Druschel, and Antony I. T. Rowstron. Defending against eclipse attacks on overlay networks. In Yolande Berbers and Miguel Castro, editors,

- Proceedings of the 11st ACM SIGOPS European Workshop, Leuven, Belgium, September 19-22, 2004*, page 21. ACM, 2004.
- [26] Atro Voutilainen. Part-of-speech tagging. *The Oxford handbook of computational linguistics*, pages 219–232, 2003.
- [27] Qiang Wang and Min Su. Integrating blockchain technology into the energy sector—from theory of blockchain to research and application of energy blockchain. *Computer Science Review*, 37:100275, 2020.
- [28] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface’s transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.
- [29] Wenpeng Yin, Jamaal Hay, and Dan Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3912–3921. Association for Computational Linguistics, 2019.
- [30] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)*, pages 557–564. Ieee, 2017.

A. Structure of Processed Logs

```
1 {
2   "_index": "data_parsed",
3   "_type": "_doc",
4   "_id": "HkEy0n8BYv2pyDWhsjXR",
5   "_score": 1.0,
6   "_source": {
7     "@timestamp": "2022-03-20T12:48:09.287Z",
8     "type": "ERROR",
9     "message": "[31m2022-03-20 12:48:09.287 UTC 007d ERRO [0m [core.comm] [31;1
10      mServerHandshake [0m -> Server TLS handshake failed in 607.238772ms with error
11      server=Orderer remoteaddress=167.94.138.46:45236",
12     "sentiment": "very negative",
13     "id_log": "MQhep38BYv2pyDWh9xpE",
14     "container": {
15       "id": "0c06a22e0a5f43b7d2ef9b6bfbfa227ae828a3fd2be0e1ab44e3f46926106640",
16       "name": "orderer.example.com",
17       "image": {
18         "name": "hyperledger/fabric-orderer:2.4.2"
19       }
20     },
21     "keywords": [
22       [
23         "1mserverhandshake",
24         "remoteaddress",
25         "failed"
26       ]
27     ],
28     "classification_labels": [
29       [
30         "Communication",
31         "Security",
32         "Connection"
33       ]
34     ],
35     "ip": [
36       "167.94.138.46:45236"
37     ],
38     "name_image_doc": {
39       "name": "hyperledger/fabric-orderer:2.4.2"
40     },
41     "stream": "stderr"
42   }
43 }
```

Listing 1: Simplified structure of processed logs used on Hyperledger Fabric network

A Provably-Unforgeable Threshold Schnorr Signature With an Offline Recovery Party

Michele Battagliola^{1,*†}, Alessio Galli^{1,†}, Riccardo Longo^{1,†} and Alessio Meneghetti^{1,†}

¹Department of Mathematics, University Of Trento, 38123 Povo, Trento, Italy

Abstract

The increase in the interest in cryptocurrencies, and the consequent need for technological maturity of blockchain-based platforms, has been the fuel for some recent advances in cryptographic research. In this context, digital signature protocols have a central role since they guarantee ownership and control of digital assets.

The absence of trusted central authorities in public blockchains, which is the very foundation of this technology, poses some interesting challenges on the management of digital identities. In particular, the computational infeasibility of restoring a lost key is a threat to anyone possessing this kind of digital assets. A possible solution to this problem is to use threshold multi-signatures, partially relying on a recovery-party whose only role, even though of paramount importance, is to intervene in case of key loss.

We present a Schnorr multi-party digital signature scheme that supports an offline participant during the key-generation phase, without relying on a trusted third party. Under standard assumptions we prove our scheme secure against adaptive malicious adversaries and capable of achieving the resiliency of the recovery in the presence of a malicious party.

Keywords

94A60 cryptography, 12E20 finite fields, 14H52 elliptic curves, 94A62 authentication and secret sharing, 68W40 analysis of algorithms.

1. Introduction

Custody of cryptocurrencies, and in general of crypto-assets, is at the very core of the burgeoning digital-asset market. Ownership is guaranteed by digital signatures and making them available and usable by the general public presents many issues: to provide a few examples, in case of inheritance heirs cannot access the crypto-assets unless they already have access to the private key, and, in general, private keys can be easily lost or forgotten, leading to the inaccessibility of the related assets. Many solutions have been devised to mitigate these problems and to enable safe custody. Some rely on

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

†These authors contributed equally.


✉ battagliola.michele@gmail.com (M. Battagliola); alessio.galli014@gmail.com (A. Galli);

riccardolongomath@gmail.com (R. Longo); alessio.meneghetti@unitn.it (A. Meneghetti)

🆔 0000-0002-8269-2148 (M. Battagliola); 0000-0003-2104-7871 (A. Galli); 0000-0002-8739-3091

(R. Longo); 0000-0002-5159-7252 (A. Meneghetti)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

personal efforts (e.g., cold storage), others simply delegate full control of the assets to a third party. Unfortunately, these kinds of solutions are partial: most either sacrifice usability or completely rely on the trustworthiness of a third party. An alternative and viable solution is to use threshold digital signatures [8]. This kind of technique addresses more comprehensively the problems above. It relies on multiple private keys, instead of a single one, which are distributed among parties, and a subset of them are required to control the crypto-assets. This approach is resilient with respect to the unavailability or loss of one party. In particular we design a three-parties protocol, that allows users to distribute their key to a custodian and a third party, like a bank or another financial institute. Security is guaranteed as long as the two helping parties do not collude, i.e., it is sufficient that one of the two remains honest to preserve the safety of the system. Furthermore, this solution is effectively agnostic to the underlying blockchain, i.e., it does not have to be supported by special features.

Starting from the highly influential work of Gennaro et al [14], several authors proposed both novel schemes [7, 19, 20] and improvements to existing protocols [4, 9, 10, 12, 13, 17, 18, 21].

Recently, in [1] and [2] the authors propose an ECDSA-compatible and an EdDSA-compatible (2,3)-threshold multi-signature protocol in which one of the users plays the role of the recovery party: a user involved only once in a preliminary setup prior even to the key-generation step of the protocol.

In this paper we propose a third, Schnorr-based, variant of [2]. The Schnorr signature algorithm has recently gained popularity in the world of cryptocurrencies, especially since its addition to Bitcoin with BIP340¹. Schnorr signatures have many advantages, such as linearity, non-malleability and provable security. In particular, they are strongly unforgeable under chosen message attacks: in the random oracle model assuming the hardness of the discrete logarithm problem, in the generic group model assuming variants of preimage and second preimage resistance of the used hash function. In contrast, the best known results for the provable security of ECDSA rely on stronger assumptions. Moreover, the threshold version presented here allows for fast computation with fewer rounds of communication with respect to ECDSA, and unlike EdDSA does not require expensive computation to derive a deterministic nonce.

We prove the protocol secure against adaptive adversaries by reducing it to the classical Schnorr scheme, assuming the security of a non-malleable commitment scheme, and an IND-CPA encryption scheme. Moreover we make some considerations about the resiliency of the recovery, an interesting aspect due to the presence of an offline party, analyzing possible changes that allow us to achieve this higher level of security.

2. Preliminaries

In this section we present some preliminary definitions and primitives that will be used in the protocol and its proof of security.

¹see <https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki>

Notation We use the symbol $\|$ to indicate the concatenation of bit-strings. Sometimes we slightly abuse the notation and concatenate a bit-string M with a group element \mathcal{P} , in those cases we assume that there has been fixed an encoding φ that maps group elements into bit-strings, so $M\|\mathcal{P} := M\|\varphi(\mathcal{P})$.

In the following when we say that an algorithm is efficient we mean that it runs in (expected) polynomial time in the size of the input, possibly using a random source.

We use a blackboard-bold font to indicate algebraic structure (i.e. sets, groups, rings, fields), a calligraphic font will generally denote elements of a finite group.

2.1. Cryptographic Hash Functions

In the Schnorr scheme (and therefore in our threshold protocol) a cryptographic hash function H is used as a Pseudo-Random Number Generator (PRNG), employed to derive secret scalars and nonces. The requirements needed for the hash function used in Schnorr signatures are analyzed in [23].

2.1.1. Schnorr Signature

Schnorr's digital signature algorithm is an efficient algorithm able to generate short signatures without sacrificing security. It is one of the first signatures that bases its security on the difficulty of discrete logarithm problem [24].

If Alice wants to send a signed message to Bob, she has to choose group \mathbb{G} with generator g of prime order q where the discrete logarithm problem is considered to be hard and a cryptographic hash function H . Then they can do the following:

1. Key Generation: Alice chooses randomly a private key $x \in \mathbb{Z}_q^*$ and computes the public key $y = g^x$;
2. Signature Generation: to sign a message m , Alice performs the following:
 - a) Choose randomly $k \in \mathbb{Z}_q^*$;
 - b) Compute $r = g^k$;
 - c) Compute $e = H(m\|r)$;
 - d) Compute $s = (k - xe)$;
 - e) The signature is the pair (e, s) .
3. Signature Verification: to verify the signature after receiving m and (e, s) , Bob performs the following:
 - a) Compute $r_v = g^s y^e$;
 - b) Compute $e_v = H(m\|r_v)$;
 - c) The signature is valid only if $e_v = e$.

2.2. Encryption Scheme

In our protocol we need an asymmetric encryption scheme to communicate with the offline party. The minimum requirement we ask for our protocol to be secure is that the encryption scheme chosen by the offline party has the property of IND-CPA [3, 22].

This hypothesis will be enough to prove the unforgeability of the protocol, but it is possible to achieve a higher notion of security by using a more sophisticated encryption scheme that supports Zero-Knowledge Proofs for the Discrete Logarithm. This will be more clearly explained in Section 4.5.

2.3. Commitment Schemes

A commitment scheme [5] is composed by two algorithms:

- $\text{Com}(M) : \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$: takes in input the value M to commit² and, using a random source, outputs the commitment string C and the decommitment string D .
- $\text{Ver}(C, D) : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \cup \{\perp\}$: takes the commitment and decommitment strings C, D and outputs the originally committed value M if the input pair is valid, \perp otherwise³.

We require a commitment scheme to have the following properties:

- Correctness: for every value M it holds $\text{Ver}(\text{Com}(M)) = M$.
- Binding: for every commitment string C it is infeasible to find $M \neq M'$ and $D \neq D'$ such that $\text{Ver}(C, D) = M$ and $\text{Ver}(C, D') = M'$ with both $M, M' \neq \perp$.
- Hiding: Let $(C, D) = \text{Com}(M_b)$ with $b \in \{0, 1\}$, $M_1 \neq M_0$, then it is infeasible for an attacker that may choose $M_0 \neq M_1$ and sees only C , to correctly guess b with more than negligible advantage.
- Non Malleability: Given $C = \text{Com}(M)$, it is infeasible for an adversary to produce another commitment string C' such that after seeing D such that $\text{Ver}(C, D) = M$, can find a decommit string D' such that $\text{Ver}(C', D') = M'$ with M' related to M , that is can only create commitments to values that are independent from M .

2.4. Zero-Knowledge Proofs

In the protocol, various Zero-Knowledge Proofs (ZKP) [16] are used to enforce the respect of the passages prescribed by the specifications. In fact, in the proof of security we can exploit the soundness of these sub-protocols to extract valuable information from the

²In the protocol and the simulations we implicitly encode every value we need to commit into a bit-string, assuming there is a standard encoding understood by all parties

³Again, in the protocol we implicitly decode valid decommitment outputs (i.e. $\neq \perp$) into the original value, assuming that the decoding is also standard and understood by all parties

adversary, and their zero-knowledge property to simulate correct executions even without knowing some secrets. We can do so because we see the adversary as a (black-box) algorithm that we can call on arbitrary input, and crucially we have the faculty of rewinding its execution.

In particular we use ZKP of Knowledge (ZKPoK) to guarantee the usage of secret values that properly correspond to the public counterpart, specifically the Schnorr protocol for discrete logarithms, and its variant that proves that two public values are linked to the same secret (see [24, 27]). The soundness property of a ZKPoK guarantees that the adversary must know the secret input, and appropriate rewinds and manipulations of the adversary’s execution during the proof allows us to extract those secrets and use them in the simulation. Conversely exploiting the zero-knowledge property we can trick the adversary in believing that we know our secrets even if we do not, thus we still obtain a correct simulation of our protocol from the adversary’s point of view.

2.5. Feldman-VSS

Feldman’s VSS scheme [11] is a verifiable secret sharing scheme built on top of Shamir’s scheme [26]. A secret sharing scheme is verifiable if auxiliary information is included, that allows players to verify the consistency of their shares. We use a simplified version of Feldman’s protocol: if the verification fails the protocol does not attempt to recover excluding malicious participants, instead it aborts altogether. In a sense we consider somewhat honest participants, for this reason we do not need stronger schemes such as [15, 25].

The scheme works as follows:

1. A cyclic group \mathbb{G} of prime order q is chosen, as well as a generator $g \in \mathbb{G}$. The group \mathbb{G} must be chosen such that the discrete logarithm is hard to compute.
2. The dealer computes a random polynomial P of degree t with coefficients in \mathbb{Z}_q , such that $P(0) = s$ where $s \in \mathbb{Z}_q$ is the secret to be shared.
3. Each of the n share holders receive a value $P(i) \in \mathbb{Z}_q$. So far, this is exactly Shamir’s scheme.
4. To make these shares verifiable, the dealer distributes commitments to the coefficients of P . Let $P(X) = s + \sum_{i=1}^n a_i X^i$, then the commitments are $\mathcal{C}_0 = g^s$ and $\mathcal{C}_i = g^{a_i}$ for $i \in \{1, \dots, n\}$.
5. Any party can verify its share in the following way: let α be the share received by the i -th party, then it can check if $\alpha = P(i)$ by verifying if the following equality holds:

$$g^\alpha = \prod_{j=0}^t (\mathcal{C}_j)^{i^j} = g^s \cdot g^{\sum_{j=1}^t a_j (i^j)} = g^{s + \sum_{j=1}^t a_j (i^j)} = g^{P(i)}.$$

3. Threshold Schnorr Signature

In this section we describe the main protocol: a $(2, 3)$ -threshold variant of Schnorr digital signature algorithm with an offline participant. Let P_1, P_2, P_3 the parties involved in the protocol, as already mentioned the goal is to allow to one of them, namely P_3 to remain offline during the key generation phase. Moreover our goal is to allow for a trustless setup, where the parties does not have to rely to a third trusted party to generate the credential. From now on we refer to P_3 as the offline or recovery party, since its role is to take part in the signing protocol if for any reason one of the two is no more able (secret key loss, unreachability, etc.).

The protocol is dividend into four algorithms:

1. Setup Phase (3.1): in this phase all three players interact to set some common parameters. Note that in a practical implementation this phase can be performed ahead of time without any real communication, because these parameters are usually fixed (e.g. for Bitcoin applications which have to use secp256k1 and SHA-256).
2. Key-Generation (3.2): performed by P_1 and P_2 to create the public key for the signature scheme and the private shards for themselves and P_3 ;
3. Ordinary Signature (3.3): used whenever P_1 and P_2 want to perform a signature. It is called ordinary signature as this should be the standard signing procedure;
4. Recovery Signature (3.4): ideally, this algorithm is executed when either P_1 or P_2 is no more able to sign. P_3 steps in and performs a signature with the remaining party. It is important to emphasize that the final signature is still a standard one, same as the one generated in an ordinary signature and indistinguishable to one obtained in the centralized protocol.

From now on “ P_i does something” means that both P_1 and P_2 perform that action. Also by saying “ P_i sends a message to P_j ” means that P_1 sends data to P_2 and viceversa.

3.1. Setup Phase

The aim of this phase is to make P_1 and P_2 agree on all the parameters required in the protocol and set up the private/public key pair used to contact P_3 in case of need.

Player 1 and 2	Player 3
Input:	Input:
Private Output:	Private Output: sk_3
Public Output: G, g, q, H	Public Output: pk_3

P_3 picks a key pair (pk_3, sk_3) for a suitable asymmetric encryption algorithm. Then P_1 and P_2 agree on a secure hash function H whose outputs can be interpreted as elements of \mathbb{Z}_q and a group G with generator g of prime order q in which the discrete logarithm problem is considered to be hard.

3.2. Key generation

This part of the protocol is performed by P_1 and P_2 to produce a common public key \mathcal{A} and to distribute shards of the corresponding private key to each player.

Player 1		Player 2	
Input:	pk_3	Input:	pk_3
Private Output:	ω_1	Private Output:	ω_2
Public Output:	$rec_{1,3}, rec_{2,3}, \mathcal{A}$	Public Output:	$rec_{1,3}, rec_{2,3}, \mathcal{A}$

1. Secret key generation and communication:
 - a) P_i randomly chooses $a_i, y_{3,i}, m_i \in \mathbb{Z}_q$ and sets $\mathcal{A}_i = g^{a_i}, \mathcal{Y}_{3,i} = g^{y_{3,i}}$;
 - b) P_i computes $[KGC_i, KGD_i] = \text{Com}((\mathcal{A}_i, \mathcal{Y}_{3,i}))$;
 - c) P_i sends KGC_i to P_j ;
 - d) P_i sends KGD_i to P_j ;
 - e) P_i gets $((\mathcal{A}_i, \mathcal{Y}_{3,i})) = \text{Ver}(KGC_j, KGD_j)$.
2. Feldman VSS and generation of P_3 data:
 - a) P_i sets $f_i(x) = a_i + m_i x$ and computes $y_{i,1}, y_{i,2}, y_{i,3}$ where $y_{i,j} = f_i(j)$;
 - b) P_i encrypts $y_{i,3}, y_{3,i}$ with pk_3 and obtains $rec_{i,3}$;
 - c) P_i sends $y_{i,j}$ and $rec_{i,3}$ to P_j ;
 - d) If the asymmetric encryption algorithm supports DLOG verification, the encryption $rec_{i,3}$ is accompanied by two NIZKPs: the first one proves that the first ciphertext in $rec_{i,3}$ is the encryption of the DLOG of $\mathcal{Y}_{i,3} = \mathcal{A}_i \cdot (\mathcal{M}_i)^3$ (where $\mathcal{M}_i = g^{m_i}$ is sent during the Feldman-VSS protocol), the second NIZKP proves that the second ciphertext is the encryption of the DLOG of $\mathcal{Y}_{3,i}$. P_i checks the NIZKPs attached to $rec_{j,3}$.
 - e) P_i checks, as in the Feldman-VSS protocol, the integrity and consistency of the shards $y_{j,i}$;
 - f) P_i computes $x_i = y_{1,i} + y_{2,i} + y_{3,i}$.
3. P_i proves in ZK the knowledge of x_i using Schnorr's protocol.
4. Public key and shards generation:
 - a) the public key is $\mathcal{A} = \prod_{i=1}^3 \mathcal{A}_i$, where $\mathcal{A}_3 = (\mathcal{Y}_{3,1})^2 / \mathcal{Y}_{3,2}$ so that $a_3 = 2y_{3,1} - y_{3,2}$. From now on we will set $a = \sum_{i=1}^3 a_i$ and we have $g^a = \mathcal{A}$;
 - b) P_1 computes $\omega_1 = 2x_1$, while P_2 computes $\omega_2 = -x_2$. Note that $\omega_1 + \omega_2 = a$;

3.3. Signature Algorithm

This algorithm is the general signature scheme in which two players, P_A and P_B , want to sign a message. Each of P_1, P_2, P_3 can take the role of either P_A or P_B depending on the situation, we call Ordinary Signature the case in which P_1 takes the role of P_A and P_2 takes the role of P_B .

Let M be the message, the parameters involved are:

Player A	Player B
Input: M, ω_A, \mathcal{A}	Input: M, ω_B, \mathcal{A}
Public Output: (s, e)	Public Output: (s, e)

The protocol proceeds as follows.

1. Generation of r :
 - a) P_i randomly chooses $k_i \in \mathbb{G}$;
 - b) P_i computes $r_i = g^{k_i}$;
 - c) P_i computes $[\text{KGC}_i, \text{KGD}_i] = \text{Com}(r_i)$ and sends KGC_i ;
 - d) P_i sends KGD_i ;
 - e) P_i computes $r_j = \text{Ver}([\text{KGC}_j, \text{KGD}_j])$;
 - f) P_i computes $r = r_A r_B$.
2. Generation of s :
 - a) P_i computes $e = H(r||M)$ and $s_i = k_i - \omega_i e$;
 - b) P_i computes $[\text{KGC}'_i, \text{KGD}'_i] = \text{Com}(s_i)$ and sends KGC'_i ;
 - c) P_i sends KGD'_i ;
 - d) P_i computes $s_j = \text{Ver}([\text{KGC}'_j, \text{KGD}'_j])$;
 - e) P_i computes $s = s_A + s_B$.
3. P_i computes $r_v = g^s \mathcal{A}^e$ and checks that $H(r_v||M) = e$.

The output signature is (s, e) . If a check fails, the protocol aborts.

3.4. Recovery Signature

This is the scenario where one between P_1 or P_2 is unable to sign. P_3 has to come back online and perform a recovery signature with the other online party. There are two different situations, depending whether the other party is P_1 or P_2 .

Firstly we consider the case where P_2 is offline and P_1 and P_3 want to perform a signature. The parameters involved are:

Player 1	Player 3
Input: $M, \omega_1, \mathcal{A}, \text{rec}_{1,3}, \text{rec}_{2,3}$	Input: M, sk_3
Public Output: (s, e)	Public Output: (s, e)

The protocol is the following.

1. Communication:
 - a) P_1 contacts P_3 and sends $\mathcal{A}, \text{rec}_{1,3}, \text{rec}_{2,3}$;
 - b) P_3 decrypts $\text{rec}_{1,3}, \text{rec}_{2,3}$ using its private key to obtain $y_{1,3}, y_{3,1}, y_{2,3}, y_{3,2}$;
 - c) P_3 computes $a_3 = 2y_{3,1} - y_{3,2}$ and $\mathcal{A}_3 = g^{a_3}$.
2. P_3 's key creation:
 - a) P_3 computes $x_3 = y_{1,3} + y_{2,3} + 2y_{3,2} - y_{3,1}$;
 - b) P_i proves in ZK the knowledge of x_i using Schnorr's protocol ($x_1 = \frac{1}{2}\omega_1$).
3. Signature generation:
 - a) P_1 computes $\tilde{\omega}_1 = \frac{3}{4}\omega_1$;
 - b) P_3 computes $\omega_3 = -\frac{1}{2}x_3$;
 - c) P_1 and P_3 perform the signature algorithm with $P_A = P_1, P_B = P_3$ using $\omega_A = \tilde{\omega}_1$ and $\omega_B = \omega_3$. Note that it still holds that $\omega_A + \omega_B = a$.

The other scenario is the one in which P_1 is offline and P_2 signs the message with P_3 :

Player 2	Player 3
Input: $M, \omega_2, \mathcal{A}, \text{rec}_{1,3}, \text{rec}_{2,3}$	Input: M, sk_3
Public Output: (s, e)	Public Output: (s, e)

The first two steps are the same as in the previous scenario, except that in the ZKP in [2b] we now have $x_2 = -\omega_2$.

3. Signature generation:
 - a) P_2 computes $\tilde{\omega}_2 = -3\omega_2$;
 - b) P_3 computes $\omega_3 = -2x_3$;
 - c) P_2 and P_3 perform the signature algorithm with $P_A = P_2, P_B = P_3$ using $\omega_A = \tilde{\omega}_2$ and $\omega_B = \omega_3$. Note that also here $\omega_A + \omega_B = a$.

4. Security Proof

In this section we discuss the security of the scheme in terms of the unforgeability properties defined below. We also discuss other security aspects, such as recovery resiliency in the subsequent Section 4.5.

Definition 4.1 (Unforgeability). A (t, n) -threshold signature scheme is unforgeable if no malicious adversary who corrupts at most $t - 1$ players can produce the signature on a new message m with non-negligible probability, given the view of the threshold sign on input messages m_1, \dots, m_k (adaptively chosen by the adversary), as well as the signatures on those messages.

The unforgeability of our protocol is formally stated in the following theorem:

Theorem 4.1. Assuming that:

- the Schnorr signature scheme instantiated on the group G of prime order q with the hash function H is unforgeable;
- Com, Ver is a non-malleable commitment scheme;
- the Decisional Diffie-Hellman Assumption holds;
- the encryption algorithm used by P_3 is IND-CPA;

our threshold protocol is unforgeable.

In Section 4.4 we will prove the theorem by showing that if there is an adversary able to forge a signature for the threshold scheme with non negligible probability $\epsilon > \lambda^{-c}$ with λ a polynomial and $c > 0$, then it is possible to build a forger \mathcal{F} that forges a signature for the centralized Schnorr scheme also with non negligible probability. The simulation works by having an oracle that feeds inputs for the centralized scheme to \mathcal{F} , our goal is to respond by generating a signature exploiting . First, it has to simulate the key generation protocol in order to match the key received from the oracle, then it can proceed with the signature part. The core of this setup is that if is able to crack our protocol, \mathcal{F} will take advantage of that and will also create a forgery for the centralized version of the oracle.

Following the definition of unforgeability, will control one player while \mathcal{F} controls the remaining two. We must consider two different scenarios: one where controls P_1 or P_2 , and the case where controls P_3 . First, we suppose without loss of generality that controls P_2 .

The adversary interacts by first participating in the key generation part to generate a public key \mathcal{A} , then starts requesting signatures on some messages m_1, \dots, m_l . Here it can either take part in the process or let P_1 and P_3 generate the signature. Eventually outputs a message $m \neq m_i \forall i$ and its valid signature with probability at least ϵ , where this is taken over the random tapes of the adversary and the honest player, respectively τ and τ_i . So we can write that

$$\mathbb{P}_{\tau_i, \tau, \mathcal{A}}(\text{ }(\tau)_{P_i(\tau_i)} = \text{forgery}) \geq \epsilon, \quad (1)$$

where $(\tau)_{P_i(\tau_i)}$ is the output of at the end of this process and $\mathbb{P}_{\tau_i, \tau, \mathcal{A}}$ denotes that the probability is taken over the random tape τ_i and the adversary tape τ .

We say that a random tape is good if

$$\mathbb{P}_{\tau_i}(\text{ }(\tau)_{P_i(\tau_i)} = \text{forgery}) \geq \frac{\epsilon}{2}. \quad (2)$$

We recall the following useful lemma, stated and proved in [2].

Lemma 4.1. If τ is chosen uniformly at random, the probability that τ is good is at least $\frac{\epsilon}{2}$.

4.1. Key generation simulation

Now we see into details how the key generation phase is simulated. \mathcal{F} receives from the challenger the public key \mathcal{A}_c for the centralized Schnorr protocol and a public key pk_3 for the asymmetric encryption scheme. The simulation proceeds as follows:

1. P_i selects random values $a_i, y_{3,i}, m_i \in \mathbb{Z}_q$ and computes $\mathcal{A}_i = g^{a_i}, \mathcal{Y}_{3,i} = g^{y_{3,i}}$;
2. P_i computes the commitment $[\text{KGC}_i, \text{KGD}_i] = \text{Com}(\mathcal{A}_i, \mathcal{Y}_{3,i})$;
3. P_i sends KGC_i , then, after receiving KGC_j , P_i sends KGD_i ;
4. P_i gets $(\mathcal{A}_i, \mathcal{Y}_{3,i}) = \text{Ver}(\text{KGC}_j, \text{KGD}_j)$;
5. Now \mathcal{F} knows all the parameters needed in the computation of \mathcal{A} , so it rewinds to step 3, aiming to get $\mathcal{A} = \mathcal{A}_c$;
6. \mathcal{F} computes $\hat{\mathcal{A}} = \frac{\mathcal{A}_c}{\mathcal{A}_2 \mathcal{A}_3}$, computes the commitment $[\text{KGC}\hat{1}, \text{KGD}\hat{1}] = \text{Com}(\hat{\mathcal{A}}, \mathcal{Y}_{3,1})$, and sends it to \mathcal{P}_1 , so that it will receive $\hat{\mathcal{A}}$ as \mathcal{A}_1 which leads to $\mathcal{A} = \mathcal{A}_c$;
7. \mathcal{F} simulates a fake Feldman-VSS with \mathcal{A} (see e.g. [2]) since it cannot compute the polynomial $f(x)$: it selects $y_{1,2}, y_{1,3}$ randomly and computes $c_{1,j} = \frac{1}{j} (g^{y_{1,j}} / \hat{\mathcal{A}})$.
8. P_i encrypts $y_{1,3}$ and $y_{3,i}$ with pk_3 , getting $\text{rec}_{i,3}$, then sends $y_{i,j}, \text{rec}_{i,3}$;
9. P_i computes x_i . Since \mathcal{F} does not know the discrete logarithm of $\hat{\mathcal{A}}$, it sets x_1 randomly;
10. \mathcal{F} participates in the ZK proofs rewinding \mathcal{P}_1 and selecting appropriate challenges in order to extract x_2 from \mathcal{P}_1 ;
11. P_j can compute the key \mathcal{A} as described in the enrollment phase. \mathcal{P}_2 can also compute ω_2 , while \mathcal{F} cannot, since it does not know x_1 .

Note that at the end of the protocol, \mathcal{F} does not know x_1 nor ω_1 , but \mathcal{F} will still be able to complete correctly the signing part by querying the oracle.

The proof of the correctness of the simulation is stated in the following lemmas. The proofs are trivial and use the same argument of the one presented in [2].

Lemma 4.2. If the Decisional Diffie-Hellman assumption holds, and the encryption algorithm used by P_3 is IND-CPA, then the simulation terminates in expected polynomial time and is indistinguishable from the real protocol.

Lemma 4.3. For a polynomial number of inputs the simulation terminates with output \mathcal{A}_c except with negligible probability.

Observation 1. It is important that in step 3 the adversary sends KGC_2 and KGD_2 before \mathcal{F} , so that after the rewinding \mathcal{A} cannot change its commitment (note that this applies also to the simulation in Section 4.2). If the order were inverted, \mathcal{A} could also use the commitment of \mathcal{F} to generate its value. Assuming the non-malleability property, \mathcal{A} does not deduce anything about the content of the commitment, but it could still use it as a seed for a random generator. If this were to happen, \mathcal{F} can guess $\hat{\mathcal{A}}$ with probability $\frac{1}{q}$ with q the size of the group, making the expected time exponential.

It is possible to swap the order in the commitment step using an equivocable commitment scheme with a secret trapdoor. In this case we only need to rewind at the decommitment step and change KCD_1 in order to match $\hat{\mathcal{A}}$.

4.2. Signature generation simulation

After the the key generation, \mathcal{F} has to deal with the signature requests issued by \mathcal{A} . When \mathcal{A} asks for a signature, \mathcal{F} performs a simulation while having access to the signing oracle that uses the previously created public key. Here \mathcal{F} can fully predict what \mathcal{A} will output and, while it does not know any secret key of P_1 , it knows everything of P_2 since all the secret values were extracted from \mathcal{A} during the ZK proofs.

The simulation proceeds as follows:

1. \mathcal{A} chooses a message m to sign;
2. \mathcal{F} queries its signing oracle for a signature for m corresponding to the public key \mathcal{A} and gets (s_f, e_f) ;
3. P_i randomly chooses $k_i \in \mathbb{Z}_q^*$, then computes $r_i = g^{k_i}$ and $[\text{KGC}_i, \text{KGD}_i] = \text{Com}(r_i)$;
4. P_i sends KGC_i , then, after receiving KGC_j , sends KGD_i and gets $r_i = \text{Ver}([\text{KGC}_i, \text{KGD}_i])$;
5. \mathcal{F} rewinds \mathcal{A} to step 4;
6. \mathcal{F} computes $\hat{r}_1 = \frac{r_f}{r_2}$, then its commitment $[\text{KGC}'_1, \text{KGD}'_1] = \text{Com}(\hat{r}_1)$ and sends KGC'_1 to \mathcal{A} so it receives \hat{r}_1 as r_1 which leads to $r = r_f$;
7. P_i computes $r = r_1 r_2$, $e = H(r||m)$, and $s_i = k_i - \omega_i e$ (\mathcal{F} picks s_1 at random);
8. P_i computes $[\text{KGC}'_i, \text{KGD}'_i] = \text{Com}(s_i)$, then sends KGC'_i ;
9. P_i sends KGD'_i and gets $s_i = \text{Ver}([\text{KGC}'_i, \text{KGD}'_i])$;
10. \mathcal{F} computes $r'_2 = g^{s_2} \cdot g^{-e\omega_2}$, if $r_2 = r'_2$ it rewinds \mathcal{A} to step 8, otherwise it sends s_1 and aborts;
11. \mathcal{F} computes $\hat{s}_1 = s_f - s_2$ with its commitment $[\text{KGC}'_1, \text{KGD}'_1] = \text{Com}(\hat{s}_1)$ and sends KGC'_1 to \mathcal{A} so it receives \hat{s}_1 as s_1 which leads to $s = s_f$;
12. P_i computes $s = s_1 + s_2$ and $r_v = g^s \mathcal{A}^e$, then checks that $H(r_v||m) = e$. If a check fails the protocol aborts, otherwise the signature is (s, e) .

Lemma 4.4. If Com is a secure non-malleable commitment scheme, the protocol above is a perfect simulation of the centralized one and terminates correctly with output (s_f, e_f) .

Proof. The simulation is identical to the real protocol except that here \mathcal{F} does not know its secret shards. Nevertheless it is still able to retrieve the correct value from \mathcal{F} by rewinding it. As above, if the protocol terminates, by construction it will terminate with output (s_f, e_f) . If \mathcal{F} is dishonest or refuses to decommit some values, the protocol aborts. Note that the check of step 10 is introduced to preserve any abort that the adversary may cause by sending an invalid s_1 . \square

4.3. Recovery signature simulation

Since \mathcal{F} can ask both types of signature, we must also consider the case of a recovery signature. The core algorithm remains the same, so the results above still holds. Here we only need to change the setup phase during which the third player recovers its secret data. There are two scenarios: one in which \mathcal{F} controls one of P_1 or P_2 and another where it controls P_3 , which is easier, since the enrollment phase can be avoided. We will proceed in order.

Trivially, if \mathcal{F} asks for a recovery signature between the two honest parties, \mathcal{F} can simply ask its oracle and output whatever it received from the oracle. So we can limit ourselves to deal with the case where \mathcal{F} participates in the signing process.

If \mathcal{F} controls P_2 the simulation works as follows:

1. P_2 sends to P_3 $\mathcal{A}, \text{rec}_{1,3}, \text{rec}_{2,3}$. Note that some of them are random data sent by P_1 ;
2. P_3 cannot decrypt the values received in the previous step. It simulates the ZKP about x_3 and extracts the secret values from P_2 ;
3. P_2 computes $\tilde{\omega}_2 = -3\omega_2$. Note that P_3 does not have the right shards so it cannot compute its secret key;
4. They perform the signing algorithm using the simulation above. Here \mathcal{F} does not know its secret key, but it can use the signing oracle to get the signature.

If \mathcal{F} controls P_1 the only difference is in the computation of $\tilde{\omega}_1 = \frac{3}{4}\omega_1$. The last case is the one where \mathcal{F} controls P_3 . The enrollment phase is done all by \mathcal{F} so it can easily generate random shards that will be sent to P_3 during the recovery signature phase and output the public key given by the oracle. Then with the same simulation as before it can simulate the signature.

4.4. Proof of the unforgeability property

Now that we have dealt with all the possible cases we need to prove Theorem 4.1:

Proof. Let $Q < \lambda^c$ be the maximum number of signature queries that the adversary makes. In a real instance of the protocol the adversary outputs a forgery after $l < Q$ queries, either because it stops submitting queries or because the protocol aborts. As we previously proved, our simulator produces a view of the protocol that the adversary cannot distinguish from the real one, therefore \mathcal{F} will produce a forgery with the same probability as in a real execution. Then the probability of success of our forger \mathcal{F} is $\frac{\epsilon^3}{8}$ which is the product of the probability of the following independent events:

1. choosing a good random tape for \mathcal{F} , whose probability is at least $\frac{\epsilon}{2}$ as per Lemma 4.1;
2. getting a good public key, whose probability is at least $\frac{\epsilon}{2}$ as shown in Lemma 4.2 and 4.3;
3. \mathcal{F} successfully produces a forgery, whose probability is again $\frac{\epsilon}{2}$ (2).

Under the assumption on the security of the Schnorr signature scheme, the probability of success of \mathcal{F} must be negligible, which implies that ϵ must be negligible too, contradicting the hypothesis that \mathcal{F} has a non-negligible probability of forging a signature for the scheme. \square

4.5. Resilience of the recovery

In our security analysis we focused on the unforgeability of the signature, however with an offline party another security aspect is worthy of consideration: the resiliency of recovery in the presence of a malicious adversary. Of course if the offline party is malicious and unwilling to cooperate there is nothing we can do about it, however the security can be strengthened if we consider that one of the online parties may corrupt the recovery material. In this case a generic CPA asymmetric encryption scheme is not sufficient to prevent malicious behaviour, because we need a verifiable encryption scheme that allows the parties to prove that the recovery material is consistent, just like they prove that they computed the shards correctly.

In particular we need an encryption scheme that supports DLOG verification as explained in point 2d of the Key-Generation algorithm. A suitable candidate could be a variant of the CramerShoup cryptosystem presented in [6]. This algorithm is equipped with a ZKP that allows the sender to prove that the plaintext encrypted is the discrete logarithm of a public value. In particular, since the protocol is a three step ZKP with special soundness, completeness, and honest-verifier zero knowledge, it is possible to build a non-interactive ZKP using the Fiat-Shamir heuristic.

5. Conclusions

In this paper, we presented a Schnorr threshold signature with the goal of providing a reliable and efficient solution for the custody of crypto-assets, both from possible attackers and from loss due to accidents of various nature. In this sense, threshold signatures

without a trusted dealer offer a perfect solution, since the private key is never created, and they overcome the limitations of blockchains that do not have native multi-signature support. Although decentralized signature algorithms have been known for a while, we are aware of only few proposals for algorithms that are able to produce signatures indistinguishable from a standard one. The protocol described in this work is, as far as we know, the first example of Schnorr threshold multi-signature allowing the presence of an offline participant during key-generation and whose signatures are indistinguishable from Schnorr ones.

The focus of this work was to shift away from DSA-like protocols, further motivated by the recent adoption of Schnorr signatures in Bitcoin⁴. Moreover, Schnorr signatures are quite a multi-party-friendly algorithm, unlike EdDSA, since we can avoid expensive tricks to generate a deterministic nonce.

Similarly to its ECDSA and EdDSA counterparts, in order to guarantee the security of the signature itself against black-box adversaries, the protocol involves a large utilization of ZKPs, that are the main bottleneck in terms of efficiency.

Future research steps could be the generalization to (t, n) -threshold schemes with more than one offline party and the extension of our notion of security. Although our protocol is susceptible to DOS attacks on the offline party, there are many ways to overcome this apparent weakness, such as the distribution of the role of the Recovery party to multiple servers or the generalization of our scheme to more than three parties.

Acknowledgments

The core results of this paper are contained in the Master’s Thesis of the second author, supervised by the first and fourth authors.

The third and the fourth authors are members of the INdAM Research group GNSAGA. The first author acknowledges support from TIM S.p.A. through the PhD scholarship.

References

- [1] Michele Battagliola, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. A provably-unforgeable threshold eddsa with an offline recovery party. CoRR, abs/2009.01631, 2020.
- [2] Michele Battagliola, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. Threshold ecdsa with an offline recovery party. Mediterranean Journal of Mathematics, 19(1):1–29, 2022.
- [3] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography, 2005. <https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf>.
- [4] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security, 2017.
- [5] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. Journal of computer and system sciences, 37(2):156–189, 1988.

⁴See <https://github.com/taprootactivation/Taproot-Activation>, accessed 29th April 2022

- [6] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 126–144, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [7] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1769–1787, 2020.
- [8] Vincenzo Di Nicola. Custody at conio-part 3, 2020. <https://medium.com/conio/custody-at-conio-part-3-623292bc9222>.
- [9] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997. IEEE, 2018.
- [10] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066. IEEE, 2019.
- [11] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.
- [12] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194. ACM, 2018.
- [13] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016.
- [14] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 354–371. Springer, 1996.
- [15] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 295–310. Springer, 1999.
- [16] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 174–187, 1986.
- [17] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. *IACR Cryptol. ePrint Arch.*, 2019:1328, 2019.
- [18] Yehuda Lindell. Fast secure two-party ecdsa signing. In *Annual International Cryptology Conference*, pages 613–644. Springer, 2017.
- [19] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages

- 1837–1854. ACM, 2018.
- [20] Philip MacKenzie and Michael K Reiter. Two-party generation of dsa signatures. In Annual International Cryptology Conference, pages 137–154. Springer, 2001.
 - [21] Philip MacKenzie and Michael K Reiter. Two-party generation of dsa signatures. International Journal of Information Security, 2(3-4):218–239, 2004.
 - [22] Antonio Marcedone and Claudio Orlandi. Obfuscation \rightarrow (ind-cpa security \leftrightarrow circular security). In International Conference on Security and Cryptography for Networks, pages 77–90. Springer, 2014.
 - [23] Gregory Neven, Nigel P Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. Journal of Mathematical Cryptology, 3(1):69–87, 2009.
 - [24] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Conference on the Theory and Application of Cryptology, pages 239–252. Springer, 1989.
 - [25] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In Annual International Cryptology Conference, pages 148–164. Springer, 1999.
 - [26] Adi Shamir. How to share a secret. Commun. ACM, 22(11):612613, November 1979.
 - [27] Victor Shoup and Joel Alwen. Σ -Protocols Continued and Introduction to Zero Knowledge, 2007. <https://cs.nyu.edu/courses/spring07/G22.3220-001/lec3.pdf>.

Cob: A Consensus Layer Enabling Sustainable Sharding-Based Consensus Protocols

Andrea Flamini^{1,*,\dagger}, Riccardo Longo^{1,\dagger} and Alessio Meneghetti^{1,\dagger}

¹*Department of Mathematics, University of Trento
via Sommarive, 14 - 38123 Povo (Trento), Italy*

Abstract

In this paper we explore a context of application of Cob, a recently introduced Byzantine Fault Tolerant consensus protocol. Cob proves to be a leaderless consensus protocol which carries out the consensus process in parallel on each component of a list of events to be observed and recorded.

We show how Cob can be used to define a consensus layer for scalable and sustainable blockchains. This layer is used to design consensus protocols based on *sharding* as a mean to achieve scalability, and on the *fragmentation of time in time-slots*, which get assigned to nodes that are instructed to create new blocks, as a mean to reduce the amount of computation and communication necessary for the maintenance of the distributed ledger.

We explain why Cob is a viable candidate to implement such consensus layer through the introduction of an auxiliary blockchain that we name Synchronization Chain.

Keywords

Cob, blockchain, consensus, sharding, time-slot, synchronization

1. Introduction

A blockchain is a distributed ledger which allows a network of nodes to record transactions in a trusted and immutable manner. The network is generally composed of independent parties which cooperate to the maintenance of the ledger without the influence of a central authority. Clearly, since a blockchain is a distributed system, one of the most important problems is the consensus. Every blockchain is provided with a consensus algorithm which allows the network of nodes to agree on the information to record into the ledger, even in presence of malicious or faulty nodes. Since consensus protocols can be quite intensive under the computational or communication point of view (e.g. proof of work or Byzantine fault tolerant protocols, respectively), the number of transaction that can be recorded in the ledger per second in average is low if compared with the centralized counterparts. Therefore, the same negative comparison can be done regarding the costs applied to the users in terms of fees (e.g Bitcoin [13] vs Visa). This causes blockchain platforms to be incapable to grow and manage an increasing number

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

^{\dagger}These authors contributed equally.

✉ andrea.flamini.1995@gmail.com (A. Flamini); riccardolongomath@gmail.com (R. Longo);

alessio.meneghetti@unitn.it (A. Meneghetti)

ORCID 0000-0002-3872-7251 (A. Flamini); 0000-0002-8739-3091 (R. Longo); 0000-0002-5159-7252 (A. Meneghetti)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

of requests, a property which is referred to as *scalability*. In particular, we say that a platform *scales* if it can easily adapt to changes in the number of users that decide to join in, as well as in the number of transaction requests that such users perform.

1.1. Preliminaries

We now introduce two techniques which are adopted to solve the problem of intensity of computations/communications, and the problem of scalability.

1.1.1. Reduction of computations or communications

Some consensus protocols, such as EOS [4], Quadrans [1] or Takamaka [3] divide the epochs in smaller time units which we call time-slots. Before the new epoch begins, a redistribution mechanism assigns each time-slot to a node in the network. These nodes are in charge of the creation of a block and if one of them does not manage to broadcast its block in time (i.e. before the end of the time-slot), then its block gets discarded by the network. In this approach, the network first reaches a consensus on the way the time-slots must be redistributed and, after that, only the node in charge during a time-slot can produce and advertise a new block. This drastically reduces the computational consumption derived by the classical protocols based on proof of work such as Bitcoin [13] which requires that the nodes execute intensive computations without any break. Similarly, this approach reduces the burden of communications between nodes given by Byzantine fault tolerant (BFT) protocols such as Algorand [2] or HoneyBadgerBFT [12].

However, the subdivision of an epoch in preassigned time-slots, although it brings several benefits in term of energetic efficiency and platform stability, it also brings one important issue: every node in the network must have access to a common clock which specifies the beginning and the end of a time-slot, an essential tool to determine whether the node in charge created the block in time or not. The problem of accessing a common clock could be avoided by using same speed clocks and an event which triggers the end of a time-slot and the beginning of a new one¹. But, even with a common clock, there would still be the problem that a message does not reach every node in the network in the very same instant of time. Therefore, a block may be received in time by some nodes, and the same block may be received late by some other nodes if it is broadcast near the end of the time-slot. In fact, in a distributed system where the messages are broadcast via gossip, the time in which a node n receives a message does not provide very precise information about the time the other nodes have seen such message. Let λ be the message propagation time, assuming a common clock exists and T is the time when n has received the block, then, if the communication happens via gossip (which is typical in the context of permissionless blockchain networks), the other nodes will receive (or have received) the block in the time interval $[T - \lambda, T + \lambda]$. The node n does not know much more than this.

This observation opens the door to a series of vulnerabilities of the system which may allow an attacker to discard blocks legitimately created and diffused by an honest node. In fact, if there is not a third party who certifies the legitimate creation and diffusion of a block, the only

¹Some consensus protocols such as Algorand [2] assume that the nodes in the network are provided with same speed clocks and each node resets its clock every time a certificate for the new block is received. Since Algorand assumes that the certificate propagation time is below a parameter λ , the delay between two nodes is upper-bounded by λ .

feedback the network receives regarding the timing of diffusion consists only of the opinion of the nodes in charge in the following time-slots.

In Section 3 we will explain how Cob, a novel BFT consensus protocol, can substitute the third party which attests which blocks have been legitimately created in time by the right node. This protocol is executed by the network of nodes and is a *leaderless consensus protocol*, therefore the only way for a block to get certified and accepted, is to be received in time by a great majority of the nodes in the network. This is exactly what we expect to happen when a node behaves honestly, and makes it impossible for an attacker to pretend that such block was not diffused in time.

Moreover, the publication of the certificates that attest that a block have been created in time can be used as the triggering event which officially ends the current time-slot and starts the new one. This approach to the declaration of the beginning of a new period of time is similar to the one adopted by Algorand [2] to declare the beginning of the next round.

A legitimate question might be why not to use a BFT protocol to reach consensus directly on the new block instead of reaching consensus on a certificate which proves the legitimacy of the newly created block. The quick answer to this question is that reaching consensus on a certificate will let us prove the legitimacy of multiple blocks simultaneously, and delegate expensive and/or difficult checks on the individual blocks, as it will become more clear in the following sections.

1.1.2. Improving the scalability

In order to solve the scalability issues of blockchain platforms, many approaches have been proposed over the years. Some of them are the *block size increase*, the use of *off-chain state channels*, *segregated witness (SegWit)*[9], the use of *directed acyclic graphs* as in [14] and *sharding*. Among these, sharding seems to be the most promising [8, 15], a description of the main platforms adopting this technique is presented in [10].

The term sharding comes from database management, where it identifies a particular type of database partitioning, that consists in dividing large databases into smaller parts, called shards. Shards are more manageable in terms of server hosting and other aspects of database maintenance, and allow to have faster query time by diversifying the responsibility of a database structure. Similarly, when we talk about sharding in the context of blockchain platform design, we refer to an architecture which divides the “usual” chain of blocks into multiple chains called shards, which are managed in parallel by different groups of nodes. This improves throughput, since many transactions can be simultaneously validated, allowing blockchains to effectively scale for a huge number of users. Although sharding is promising, it faces many challenges that the community must efficiently and securely solve. For example, one should note that, if a network is divided in shards, for an adversary it is potentially easier to take control of a single shard compared with the whole network. In fact, its impact in terms of ratio of nodes it controls grows linearly with the number of shards the network adopts to record transactions. Another challenge the protocol designers must deal with is the inter-shard communication: nodes working on different shards might be in possess of or access to different data sources. Therefore, the protocol designer must assure that transactions elaborated by different shards

are consistent despite the fragmentation of the transaction insertion process.

Since sharding aims to maximise the scalability of a platform depending on the underlying network of nodes (with great focus on preserving the security requirements), and the network conditions can suddenly evolve², it is good practice to regularly update the configuration of the system (i.e. the actors and parameters involved). Each period of time in which the system configuration is updated is referred to as *epoch*.

In order to better comprehend what sharding is, we briefly describe some of the main components of a consensus protocol for a blockchain implementing sharding. We refer to the surveys [15, 8] for more details about blockchain sharding.

1. *Identity establishment and shard formation*: this process aims to identify the single nodes who take part to the protocol execution and (randomly) assign them to a specific shard. This process should prevent Sybil attacks from being successfully performed by malicious entities who manage to create multiple identities.
2. *Intra-shard consensus* Each node within a shard must execute a consensus protocol to reach agreement on the transactions to be recorded in the fragment of ledger which is under that shard's control. Here, we make a distinction between two possible approaches to intra-shard consensus: *weak* and *strong consistency*. According to the definition in [8], weak (or eventual) consistency means that different nodes might end up having different views of a blockchain, which leads to forks, therefore a certain number of blocks at the end of the blockchain need to be truncated to obtain stable transactions. Contrarily, strong consistency means that after the generation of a valid block, every non-faulty node shares the same view and therefore no forks can happen.
3. *Cross-shard transaction processing*: it is essential that the transactions processed by the shards are consistent not only within the shard they belong to, but also across the whole system. Therefore, for cross-shard transactions, which are transactions which involve information processed by more than a shard, a network must adopt some mechanism which allows synchronization and reconciliation of transactions processed by different groups.
4. *Epoch reconfiguration*: in order to guarantee the security of the shards, they may need to be reconfigured, requiring both reconfiguration rules (to let the platform respond to the network evolution) and possibly a randomness source.

In Section 3 we will explain why Cob can be adopted to deal with the epoch reconfiguration and can contribute to the cross-shard transaction processing in a sharding-based consensus protocol.

1.2. Contribution

Cob [6] is a novel BFT protocol (i.e. a strong consistency protocol) which is an evolution of the MBA protocol [7]. MBA is defined for complete synchronous networks, therefore it can

²The network evolution refers to: a) new nodes who decide to join the network or nodes that decide to leave it, b) nodes who decide to actively partake to the consensus process and others which decide to be passive and only have access to the information recorded into the ledger, c) nodes which become faulty over time (or, more importantly, an attacker corrupts some of them whenever it believes it is profitable).

be executed only by small networks with few dozens of nodes, while Cob can be executed by incomplete gossiping networks that can have millions of nodes, a property that makes it suitable for networks of permissionless blockchains. The aim of Cob is to allow a network of nodes to reach consensus on a list of time-stamps of events that are expected to happen in a time interval, we will explain why it can be adopted in the design of consensus protocols for blockchains implementing sharding.

In this context, Cob can be used to let the network make multiple decisions simultaneously, for example decisions about which shards have correctly performed their job, or decisions about the evaluation of parameters which characterize the protocol epochs on the basis of the network conditions. We will emphasize the advantages that Cob brings to the organization of the workload that must be executed by the shards and show some performance evaluations regarding a Cob execution under the framework we will describe.

Outline In Section 2 we provide a high-level but comprehensive description of Cob, which is fully presented in [6]. In particular we underline the properties that Cob satisfies and the assumptions it relies on. In Section 2.2 we describe its workflow and building blocks, namely the Multidimensional Graded Consensus and Multidimensional Binary Byzantine Agreement [7, 6].

In Section 3 we describe how Cob can be used to create a framework for the design of sustainable and scalable blockchain platforms. Scalability is obtained by using the sharding technique, sustainability is obtained by dividing time in time-slots, during which some prescribed nodes are expected to create blocks of processed transactions. In Section 3.3 we propose a solution to put in practice the principles previously presented.

Finally, in Section 4 we compare the performance of Cob and Algorand as consensus protocols for the solution presented in Section 3.3. The comparison is centered into quantifying the amount of data broadcast in the network by the nodes.

2. The consensus protocol Cob

In this section we provide a high-level description of *Cob*, a novel consensus protocol which efficiently solves the following problem:

Problem 2.1. *Given a set of events which a network of nodes can observe, how can the nodes reach consensus on some relevant information about such events?*

The problem above is discussed in [7, 6] by considering the presence of malicious nodes in the network, and this leads to the identification of two properties that such a consensus protocol must satisfy in order to maximise the amount of agreed-upon meaningful data.

1. The consensus protocol must be *leaderless*, which means that there is no single node proposing a protocol output and the other nodes decide whether to accept it or not, but rather several nodes must collectively construct the output list. The reason behind this is that a leader, if honest, would propose a list of relevant information which is heavily influenced by its own point of view (which in some cases might lead to incorrect decisions),

and if the leader is malicious, it may easily perform censorship attacks refusing to include some information in the list, or deliberately include invalid information. In both cases, if the network does not agree even with one component proposed by the leader, it will reject the leader’s proposal, and this process is repeated until a leader proposes a list which gets accepted by a majority of the network. Note that this might not even happen, in fact, if there is a wide disagreement among the nodes about one or more components, there might not exist a list which is accepted by such majority.

2. The consensus process must be carried out in *parallel* and independently on each component of the list, so that disagreement on a single component does not affect the consensus on the others. In this regard, if a specific component can not be agreed upon on any meaningful value due to a wide disagreement among the nodes of the network, then the network must be able to identify this network condition and manage to reach consensus on a default value that we identify with \perp .

In [7] there is a simple example which helps to understand why these two properties have such a great impact on the way consensus is achieved. In [7] it is also presented a predecessor of Cob as a solution to Problem 2.1 for a relatively small network of a fixed number n of nodes, under some strong communication assumptions, considering an attacker that controls less than $\frac{1}{3}$ of the nodes. In particular it is assumed a strongly-synchronous communication model and a complete network, where every node could instantaneously send a message to every other node. These assumptions are not practical and dramatically reduce the number of application contexts. In fact, under the *complete and synchronous network* (CS network) model, the communications between nodes happen instantaneously, via direct channels, every time a common clock (i.e. shared by all the network participants) ticks the beginning of a new protocol step. Since it is assumed that the network is complete, it is essential, for sake of efficiency, that the network is composed of a small number of nodes (which does not exceed the hundreds). The evolution of Cob presented in [6] overcomes these shortages, and can be adopted by wide networks of nodes (even with millions of participants), making it way more practical and its adoption suitable for the network of permissionless blockchains. From now on, when we refer to the protocol Cob we refer to the second definition [6], unless explicitly specified.

2.1. Cob: network and communication assumptions

Since in complete networks the number of messages exchanged through the network grows exponentially with the number of network participants, for practical applications it is more convenient to consider a network model which differs from the CS network, such as the *Asynchronous Gossiping Network* (AG network)³ presented by Micali in Algorand [2].

In this model messages are broadcast in the network in a gossiping fashion: a procedure characteristic of peer-to-peer communications where messages pass from one node to its neighbours and so on until they reach every node. In gossiping networks the network relies on

³Algorand describes the environment in which it is defined as asynchronous. This is because the communications between nodes happen via gossip and the protocol steps, which for a single user are non-overlapping time intervals, for different users may overlap due to lack of clock synchronization. However, since Algorand assumes that there is a predetermined upper-bound to the time required by a message to reach (almost) every node, and therefore there is an upper-bound to the delay between different nodes, Algorand can not be considered an asynchronous protocol.

each member to pass messages along to its neighbours, therefore it is reasonable to envisage the network as an incomplete, connected and non-directed graph. We assume that a message sent by an honest node reaches every honest node within a time limit that depends on the size of the message itself. Since malicious nodes can behave arbitrarily, the previous assumption means that they cannot be cut vertices in the network graph, that is the graph remains connected even without the edges connected to malicious nodes. We will also require that the ratio of malicious or faulty nodes is less than $\frac{1}{3}$.

2.1.1. Timing assumptions

In an AG network there does not exist a common clock (as in the case of CS network), but it is assumed that all network participants are provided with *Same-Speed Clocks* [2]. In other words, it is assumed that each network participant has its own clock and that the clocks all have the same speed, even if they are not synchronized in any way. However, it is assumed that there is an upper-bound λ on the time required by a node to diffuse in the network a "short" message. Therefore, this assumption implies that the non-synchronized clocks can reach a sort of synchronization in the following way: suppose that a node communicates to the other nodes, via a short message M , the beginning of a new protocol execution. This node will immediately reset its private clock to 0, as the broadcast of M begins, and the other nodes will do the same once they are reached by M . Since the message M reaches every node in the network within time λ , every node will reset to 0 their own private clock in the absolute time interval $[0, \lambda]$ (where 0 is the absolute time when the first node broadcasts M), causing the delay between different nodes to be upper-bounded by the parameter λ . Afterwards the time discrepancies do not vary because of the same-speed nature of the clocks.

We have explained how an AG network addresses the goal of designing a practical model which, on one hand it does not require a node to send a message to every single node every time it wants to share some information with the whole network, but on the other hand it forces the nodes to maintain their private same speed clocks slightly asynchronous (with the delay which is upper-bounded by a constant value λ).

2.1.2. Sortition mechanism

Another relevant aspect regarding the design of Cob is the following: since the nodes in the network which adopts Cob as consensus protocol can be as wide as needed, it is essential that not every node in the network broadcasts a message at the end of every step. This would clearly cause a network overload. In order to address this problem, Cob uses a *sortition mechanism* which instructs some nodes to be active during a given protocol step (i.e. to broadcast a message) while assigning to the other nodes a passive role (i.e. just collect and help broadcast the messages of the players). In order to better clear up this distinction, from now on, in a specific step, we will call *players* only the nodes selected to be active and broadcast their message, while a generic node of the network will be referred to as a *user*.

2.2. High level description of Cob

We now provide a high level description of the protocol. We first describe two protocols which are the building blocks of Cob and we explain how they achieve the core properties mentioned at the beginning of this section: the fact that Cob is leaderless, and that the consensus process is carried out in parallel on the components of the list. The first component is the *Multidimensional Graded Consensus (MGC)* and the second one is the *Multidimensional Binary Byzantine Agreement (MBBA)*. Both protocols are an extension to the multidimensional case of protocols presented by Micali and adopted in Algorand [2], namely the Graded Consensus protocol [5] and the Binary Byzantine Agreement protocol [11].

2.2.1. Cob's building blocks

The MGC is a 3 steps protocol which starts once the network observes the events they want to time-stamp and requires the players (i.e. the users who are elected to be active in a given step) of the first step to broadcast their list of observed values created during the observation phase. Once the 3 steps are executed, each node n in the network privately builds a list $\mathbf{v}^{(n)}$ of relevant information about the observed events (note that the lists can eventually be different for different nodes). Together with the list of relevant information, each node n computes a grade $g_i^{(n)} \in \{0, 1, 2\}$ associated to each component i of the list, which represents the confidence that such value is well spread around the network, according to the information received. In particular, a grade of 0 represents a high disagreement in the network; 1 represents a state of uncertainty given by an intermediate number of messages advertising the corresponding value; 2 guarantees the node which computed the grade that every honest nodes has recorded the same value in that component.

Distinct nodes might have saved different lists of relevant information, and they might have also recorded different grades, based on the messages received in the steps of MGC Protocol. However, it is proven that, for each component, the difference between the grades of honest nodes is $|g_i^{(n)} - g_i^{(m)}| \leq 1$ and it is also proven that, if $g_i^{(n)} \geq 1$ for each honest node, then the relevant information recorded in the i -th component is the same for every honest node. This implies that if some honest node sets $g_i^{(n)} = 2$ then the relevant information saved by the nodes in the i -th component is the same for every node. This is a remarkable information, but we recall that the protocol is executed by nodes that do not trust each other. Therefore, it is necessary to find a way to let the nodes who are certain that a relevant information is shared by all the honest nodes convince the nodes who are not certain about it. For this purpose the network executes the protocol MBBA: a 3 steps loop which allows the nodes in the network to reach agreement on a list of bits with the same dimension of $\mathbf{v}^{(n)}$. The scope of this protocol execution is to detect the components of the list of relevant information $\mathbf{v}^{(n)}$ which are the same for every honest node. In particular, after the MGC protocol execution, each node will build a list of bits setting each component to: 0 if it is assured that all the nodes are in agreement on that specific component (i.e. the associated grade is 2), 1 otherwise. Now the network is ready to perform the MBBA protocol, since every node has its own private initial list of bits. In [7, 6] it is proven that the MBBA is a Byzantine Agreement protocol, which allows a network of nodes provided with an initial list of bits to reach consensus on a shared list of bits \mathbf{b} . In the

context of Cob, the private initial list will be computed from the output of MGC, but at the end of the MBBA execution, the nodes will reach consensus on \mathbf{b} , which allows every node n to compute the list of relevant information (the output of Cob) on which the honest nodes can be in agreement. This list \mathbf{v} is built in the following way: for every honest node n $\mathbf{v}_i = \mathbf{v}_i^{(n)}$ if $\mathbf{b}_i = 0$ otherwise $\mathbf{v}_i = \perp$, which means that such component is left blank.

In [6, 7] it is proven that Combining MGC with MBBA it is possible to solve Problem 2.1. We omit the formal definition, and just summarize the main protocol steps of Cob using the building blocks previously described. For a detailed description of the protocol we refer to [6].

Protocol 1 Cob

- **Observation Phase** For every user u in the network:
 - u observes the events $\mathbf{E} = (E_1, \dots, E_m)$ that must be time-stamped;
 - u locally records the observed values $\mathbf{o}^{(u)} = (o_1^{(u)}, \dots, o_m^{(u)})$.
 - **Multidimensional Graded Consensus** For every user u in the network:
 - u starts the execution of MGC with input the list $\mathbf{o}^{(u)}$;
 - u takes part actively to a step of MGC only if it is elected as a player via the random sortition mechanism adopted by Cob;
 - output 1 of MGC: u locally saves the list of values $\mathbf{u} = (\Theta_1^u, \dots, \Theta_m^u)$, given by MGC;
 - output 2 of MGC: from the list of grades $\mathbf{g}^u = (g_1^u, \dots, g_m^u)$ given by MGC, u obtains a list of bits $\mathbf{v}^{u,0} = (v_1^{u,0}, \dots, v_m^{u,0})$, where $\forall i \in \{1, \dots, m\}, v_i^{u,0} = 0 \iff g_i^u = 2$, and $v_i^{u,0} = 1$ otherwise.
 - **Multidimensional Binary Byzantine Agreement** For every user u in the network:
 - u starts the execution of MBBA with input the list of bits $\mathbf{v}^{u,0}$;
 - u takes part actively to a step of MBBA only if it is elected as a player;
 - output of MBBA: u builds a certificate for $\mathbf{v}^u = (v_1^u, \dots, v_m^u) = \mathbf{v} = (v_1, \dots, v_m)$, which is the same for every honest user in the network.⁴
 - **Cob Output Determination** Being \mathbf{v} the output of MBBA and \mathbf{u} the first output of MGC computed by the user u , u computes the output of Cob $\text{out}^u = (\tilde{\Theta}_1^u, \dots, \tilde{\Theta}_m^u)$, where $\forall i \in \{1, \dots, m\}, \tilde{\Theta}_i^u = \Theta_i^u$ if $v_i = 0$ and $\tilde{\Theta}_i^u = \perp$ if $v_i = 1$.⁵
-

We underline the fact that the way MBBA is used in Cob is the same way BBA is used in Algorand: the goal is to decide whether to reject or accept a candidate piece of information (for Algorand a block, for Cob an observed value or a relevant information about a give event). However, MGC is used in a very different way: while GC in Algorand is used to determine the leader of a given protocol run, MGC in Cob is used to collect the opinion of several nodes advertising the list of values they have observed⁶.

⁴What the network is actually doing during the MBBA execution is identifying the components of the vectors \mathbf{u} which are the same for every honest user u . In particular if agreement on a component c of \mathbf{v} (the list of bits) is reached on 0, i.e. $v_c = 0$, then this means that the honest users share the same value Θ_c^u and they will preserve it, otherwise, if agreement has been achieved on 1, i.e. $v_c = 1$, this means that the network could not be convinced that the honest nodes share the same value Θ_c^u .

⁵It is proven that for each pair of honest users u_1, u_2 , $\text{out}^{u_1} = \text{out}^{u_2}$ holds.

⁶Algorand is a leader-based consensus protocol for a blockchain used to exchange cryptocurrency, therefore it tolerates that some blocks may be created by malicious nodes and contain no transactions. In fact, in many applications it is not necessary that a transaction request is immediately included in the newly created block, what is essential is that eventually an honest node will create a block which includes the pending transaction request.

3. Applicability of Cob

In this section we will explain how Cob can be used to regulate a scalable and sustainable blockchain. As we explained in Section 1.1, an approach which can be used to reduce the amount of computation or communication necessary to the maintenance of a blockchain is the subdivision of time into preassigned time-slots. This concept clearly can be combined to sharding in order to increase the scalability of a blockchain platform. What we obtain is a sharding-based blockchain in which for each shard the network assigns a time-slot to a specific node. Therefore, if the number of shards in a given epoch is m , then there must be m nodes, one for each shard, who are expected to publish a block within the end of the time-slot. But let us proceed step by step in the description of how Cob can be used to achieve this protocol structure.

3.1. Cob and time fragmentation

First, we explain how Cob can be adapted to the certification of block creation. As we have mentioned in Section 2, Cob is a leaderless consensus protocol which has been designed to let a network of nodes reach consensus on the description of a set of events which are expected to happen in a time interval. If we can consider a standard blockchain (a single shard) which is maintained with the use of preassigned time-slots, then there is a single event that the network observes during every time-slot: the creation of a block performed by the node in charge. Note that the fact that Cob carries out the consensus in parallel on each component of the list of events observed is not relevant now (since we are considering only one event), but it will become relevant in Section 3.2.

If the following statements hold:

1. nodes are provided with same speed clocks;
2. it is possible to upper-bound the diffusion time of a message of fixed weight;
3. the nodes agree on t , the duration of a time-slot;
4. the network agrees on a list L which assigns each time-slot to a given node;

then we can describe a protocol which manages the definition of time-slots and guarantees that an attacker can not pretend it has received late a legitimately created block.

In Section 3.2 we will explain how to obtain the last two items of the list, namely the duration t of the time-slots and the list L ; for what concerns the first two items, they are commonly adopted assumption in distributed protocol definitions. Assuming that each network member is in possess of the information above, the protocol could work in the following way:

Protocol 2 Time-slot

- **Synchronization setup:** the network executes an instance of Cob to decide when to start; as soon as the network creates a certificate for the message `start`, the actual protocol can begin. Since the upper-bound for the diffusion time of the certificate is λ , the delay between any two honest nodes is less than λ .
- **Block creation:** the node in charge, according to the list L builds a block of transactions and before time $t - 2\lambda$ broadcasts this block.

- **Timing evaluation:** the nodes of the network start executing Cob when their own private clock signs time t , and try to reach consensus on the digest of the newly created block.⁷
- **Certificate creation and start of a new time-slot:** each player marks the end of the current time-slot and the beginning of the new one as soon as the reception of a certificate for the newly created block (produced by the network via Cob). The nodes reset their same-speed clocks (the delay is given by the diffusion time of a certificate, which is again λ).
Return to Block creation.

In Protocol 2 it is shown that, assuming the existence of a list L , which assigns each time-slot to a node, this iterative protocol guarantees that if the right node creates and broadcasts a block in time, the network can certificate the correctness of the creation process.

Up to now, the network has not evaluated the transactions included into the block, however, this can be done right after timing verification. Assuming that L , the consensus protocol, and the semantic rules which define which transactions can be included into the ledger are public knowledge, the only aspect that can bring the nodes to disagreement is whether the legitimate node has created its block in time. The disagreement may be caused by the delay between nodes and the time of diffusion of messages. Once this information is agreed upon through Cob, every honest node will be able to determine if the transactions are invalid (therefore the block will not be taken into account) or the block can be preserved.

3.2. Cob for time-slot assignation and sharding consensus

As emphasized in Section 1.1.2, a key concept in the design of blockchains implementing sharding is the *epoch*: a time interval in which the system configuration (i.e. protocol parameters and nodes partaking the consensus process) is fixed [8]. Once the epoch ends, the actors executing the consensus protocol may be substituted and, if the network has evolved, the protocol parameters can be updated accordingly.

While explaining how Cob can be useful in the implementation of architectures based on sharding, we will follow the guidelines described in Section 3.1, maintaining the division of time in preassigned time-slots for each shard. We remind that our goal is to propose a consensus layer which may help in the creation of sustainable and scalable blockchain platforms. The first adjustment that must be done regards the list L which deals with the time-slot assignation, which must cover the time-slots of an epoch and then must be updated for the following one. Since in the sharding case there is more than one chain of blocks, the list L , together with the time-slot, must specify the shard on which a node must append its block.

Cob can be very useful in the definition of a sharding based architecture mostly by periodically making a freeze frame representing the network status and defining the next epoch configuration. Since Cob is a strong consistency protocol, once the frame describing the network status is published, the network will consider those information final, and act accordingly. For instance, Cob can be used to let the network determine the protocol parameters of an epoch on the basis of the information previously broadcast by the nodes. Examples of protocol parameters that must be agreed upon to define an epoch are the following:

⁷Note that if the creator of the block has broadcast it before $t - 2\lambda$ (according to its own time reference), then every node has received the block within time t (again, according to their own time reference), since λ upper-bounds the time for the message diffusion, and also the delay between two nodes.

1. the number of shards;
2. the number of time slots;
3. the duration of the time-slots;
4. the list L of nodes in charge of the creation of a block in a given time-slot and a given shard.

Note that these parameters must be determined according to the information observed during the previous epochs, and the consensus protocol must make explicit some deterministic rules to let the network easily reach consensus on the values. In fact we recall that consensus can be reached if there exist, at the beginning of the protocol, a sufficiently large agreement on the values proposed by the single nodes: then the consensus protocol makes this agreement explicit, reliable and final. Therefore, the evaluation of the epoch parameters can be seen as a description of events observed during the previous epochs. For example, the number of shards active during an epoch could depend on the number of nodes that apply for becoming block creators in the following epoch: the higher the number of candidates, the higher the number of shards. This means that each protocol parameter can be seen as a description of events observed in a given time interval, therefore they can be determined executing an instance of Cob.

It is necessary to clarify the fact that some parameters depend on other parameters, for example the list L depends on the number of shards and the number of time-slots. In this case it is sufficient to validate the list L to consequently implicitly fix the number of shards and the number of time-slots.

3.3. The Synchronization Chain based on Cob

Now that we have explained how Cob can be useful in the creation of a sustainable and scalable blockchain, the next question is: how can we build a framework which is agnostic of the underlying sharding consensus components (i.e. intra-shard consensus, cross-shard transaction processing, and shard formation), and put into practice the ideas described in Section 3.2 and Section 3.1?

This can be done introducing another independent chain, which we call *Synchronization Chain*, which is maintained by the network and has two main scopes:

- *synchronize the work of the nodes who work in different shards*: the Synchronization Chain dictates the beginning and the end of the time-slots and hash-links the blocks that have been legitimately created in time. This can be done in the following way: after every time-slot, the nodes working at the maintenance of the Synchronization Chain reach consensus on the set of blocks which have been created (by the nodes prescribed by L) and broadcast in time during that time-slot. The consensus is reached on the digest of these blocks, therefore a block of the Synchronization Chain contains the list of blocks created in time for each shard, and when this block gets published, the network starts the new time-slot and knows which blocks have passed the first validation (which is only about timing and legitimacy);
- *deal with epoch reconfiguration*: the block of the Synchronization Chain created in the last time-slot of each epoch, together with the hash pointers to the blocks of the shards mentioned above, contains the parameters of the following epoch. The consensus protocol

must specify how the parameter must be valued, and must specify how the nodes must create the list L . The nodes maintaining the Synchronization Chain, follow these rules to compute the parameters, and produce this larger block during the last time-slot of an epoch. With this information the network activates the next epoch and the new time-slots.

Table 1

A block of Synchronization Chain created in a time-slot of epoch h .

HEADER	
$H(S_{h,i-1})$	hash-pointer to previous Synchronization Block
DATA	
$H(B_{h,i}^s)$	
$H(B_{h,i}^1)$ ⋮ $H(B_{h,i}^m)$	hash-pointers to the blocks created during i -th time-slot of epoch h on shard s
EPOCH DATA	
parameters	(only in blocks created after the last time-slot of epoch h) value of parameters for following epochs
List L	list of nodes that will create blocks in the assigned shards in the next epoch

In this context, Cob is well-suited to be used as the consensus protocol for the Synchronization Chain. In fact, besides being leaderless, a property which guarantees the authenticity of the data agreed upon, it efficiently carries out the consensus process in parallel on each component of the list of events, which is essential in this kind of applications, as emphasized in Section 2. Since the agreement on some components of the list of observed events might be reached on \perp , the blockchain consensus protocol must determine some default values for the epoch parameters. That is, there should be a rule which decides the value of the parameters to be used when agreement is reached on \perp . For example, the configuration of the previous epoch could be maintained, otherwise the network could adopt some fixed configuration. This design choice depends on the application context.

The protocol that describes the use of the Synchronization Chain can be summarized in the following way:

Protocol 3 Synchronization Chain

- Epoch reconfiguration and timing evaluation: the network executes an instance of Cob to decide the new epoch parameters, and the blocks created in time during the current time-slot. As soon as the network reaches agreement, the block of the Synchronization Chain is published, advertising the parameters and the digest of the blocks created in time. The new epoch begins. Go to Certificate creation and start of a new time-slot.

- **Timing evaluation:** the nodes of the network start executing Cob when their own private clock signs time t , trying to reach consensus on the digest of the newly created blocks, one for each shard.
Go to `Certificate creation` and start of a new time-slot.
 - **Block creation:** the nodes in charge, according to the list L published at the end of Epoch reconfiguration and timing evaluation, build a block of transactions and before time $t - 2\lambda$ broadcast their block.
If the current time-slot is the last of the epoch, go to Epoch reconfiguration and timing evaluation, otherwise go to Timing evaluation.
 - **Certificate creation and start of a new time-slot:** as soon as a player receives a certificate for the newly created blocks (produced by the network using Cob), it marks the end of the current time-slot, and the beginning of the new time-slot. Therefore the nodes can reset their same-speed clocks (which will be delayed by at most by the diffusion time of a certificate, namely λ) and return to `Block creation`.
-

Since Cob is a consensus protocol with strong consistency, the information included in the blocks of the Synchronization Chain are final. Moreover, thanks to the fact that Cob is leaderless, the evaluated fields are trustworthy. Therefore, the Synchronization Chain can be seen as a trusted third party who communicates the outcome configuration for the following epoch based on the information the network has collected during the current (but possibly also previous epochs).

The intra-shard consensus protocol can be weak, which allows lower communication consumption, however, due to the Synchronization Chain’s timing evaluation, there is a strong consistency consensus on the legitimately created blocks. This simplifies Cross-shard transaction processing, since the nodes working in different shards know which transactions can potentially become final.

4. Performance analysis

We can now take the performance analysis of Cob included in [6] and apply it to the use case discussed here. The analysis is focused on comparing the amount of data broadcast in the network during a instance of Cob executed on a list with ℓ components with the amount of data broadcast in an execution of ℓ instances of Algorand to reach Consensus on each relevant information regarding the observed events.

In order to provide a comparison which fits the use case of the Synchronization Chain described in Section 3.3, we must identify a reasonable number of parameters necessary to perform the epoch reconfiguration, then we can vary the number of shards and time-slots to determine the number of elements in the list L .

Once this is done, we can quantify the weight of messages broadcast at the end of the last time-slot of each epoch, when the epoch reconfiguration is performed, and the weight of messages of all the other time-slots, when the network must notify only the blocks created within the end of the current time-slot. For sake of simplicity we will assume the number of time-slots during an epoch is fixed, so that we can vary only the number of shards.

The number of components Nc_e that must be agreed upon in the last time-slot of epoch e can be parameterized as follows: $Nc_e = \alpha + \beta Ns_{e+1} + Ns_e$ where Ns_x is the number of shards in

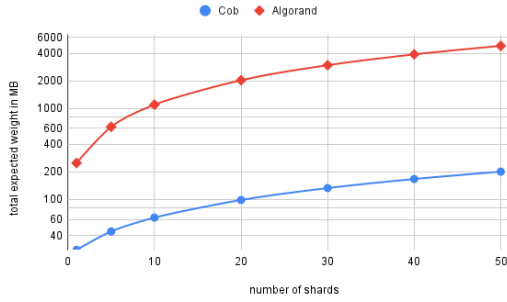


Figure 1: Amount of data broadcast in the network (in MB) using Algorand or Cob during the last time-slot of each epoch, in terms of the number of shards. Linear scale in the x-axis and logarithmic scale in the y-axis.

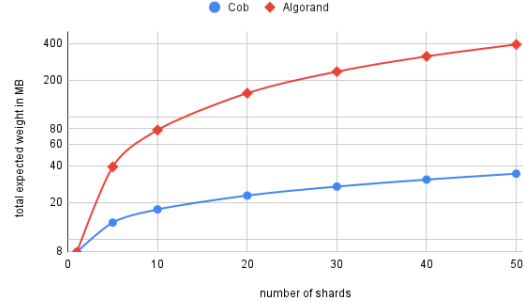


Figure 2: Amount of data broadcast in the network (in MB) using Algorand or Cob during any time-slot of each epoch but the last one, in terms of the number of shards. Linear scale in the x-axis and logarithmic scale in the y-axis.

epoch x , α is the number of parameters defining the general configuration of the platform, and β is the number of parameters specifying some properties of each shard of the next epoch (e.g. the nodes designated to work on a given time-slot).

Let us fix the number of time-slots for epoch to 10, and choose the parameters $\alpha = 20$ and $\beta = 10 + 1$ (these values are chosen in the same order of magnitude as the ones relative to the blockchain Quadrans [1], where 10 of the β parameters characterizing the shards are the nodes assigned to a given time-slot, for each shard). For what concerns the regular time-slot, the number of components that must be agreed upon is simply Ns_e , i.e. the number of blocks that should be created (note that this number is considered also in the last time-slot).

The results of the comparison are presented in Figure 1 and Figure 2, and are based on the performance analysis included in [6] for the values of parameters α and β mentioned above. For sake of simplicity we also assumed $Ns_e = Ns_{e+1}$, i.e. the number of components in the last time-slot is $20 + 12Ns_{e+1}$.

5. Conclusions

In this paper it is shown how the consensus protocol Cob, presented in [7, 6], can be useful for designing sustainable sharding-based consensus protocols for blockchains, as suggested in the original papers [7, 6]. The key concept is the following: in an architecture that pre-assigns time-slots to nodes, the node assigned to a given time-slot in a shard is common knowledge, and the network is in agreement about this. The same holds for the quality evaluation of the transactions included in the blocks: every honest node can determine whether a given block contains valid transactions according to the chain of blocks to which it is connected. In fact, the consensus protocol must define how the ledger can evolve and, given a static status of the ledger, which transactions can be appended. The only thing which remains subjective for each node is the moment in which a message is received. Someone might have received it in time, someone else might have received it late. These messages may be blocks of transactions or data

useful for the epoch reconfiguration, anyways, it is essential for the network to have a clear image of the status of the evolving system (the blockchain), in particular when the system is maintained by several groups working in parallel, which is the case of a blockchain that uses sharding to scale. We propose a solution to this problem using Cob, so that consensus can be reached on these subjective data (the network decides on the basis of what the majority of the nodes have observed) and every node in the network can have the same view on how the ledger is evolving.

Acknowledgments

The publication was created with the co-financing of the European Union - FSE-REACT-EU, PON Research and Innovation 2014-2020 DM1062 / 2021. The authors are members of the INdAM Research group GNSAGA. The first author acknowledges support from Eustema S.p.A. through the PhD scholarship.

References

- [1] Michele Battagliola, Andrea Flamini, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. *Quadrans blockchain*, 2021.
- [2] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.
- [3] Takamaka enterprise blockchain. Company oriented blockchain. *takamaka.io/whitepaper*, 2020.
- [4] D. Larimer et al. Eos.io technical white paper v2. <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>, 2017.
- [5] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.
- [6] Andrea Flamini, Riccardo Longo, and Alessio Meneghetti. Cob: a leaderless protocol for parallel byzantine agreement in incomplete networks. *Distributed and Parallel Databases*, 2022.
- [7] Andrea Flamini, Riccardo Longo, and Alessio Meneghetti. Multidimensional byzantine agreement in a synchronous setting. *Applicable Algebra in Engineering, Communication and Computing*, pages 1–19, 2022.
- [8] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, Zongyang Zhang, Tong Li, Bin Hu, Fritz Henglein, and Rongxing Lu. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *CoRR*, abs/2102.13364, 2021.
- [9] Eric Lombrozo, Johnson Lau, and Pieter Wuille. Segregated witness (consensus layer). *Bitcoin Core Develop. Team, Tech. Rep. BIP*, 141, 2015.
- [10] Alessio Meneghetti, Tommaso Parise, Massimiliano Sala, and Daniele Taufer. A survey on efficient parallelization of blockchain-based smart contracts. *Annals of Emerging Technologies in Computing (AETiC)*, 3(5), 2019.
- [11] Silvio Micali. Byzantine agreement, made trivial, 2016.
- [12] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger

- of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.
- [13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008.
- [14] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. Cryptology ePrint Archive, Report 2016/1159, 2016. <https://ia.cr/2016/1159>.
- [15] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61, 2019.

Olive Oil as Case Study for the *-Chain Platform

Stefano Bistarelli¹, Francesco Faloci^{2,3,*}, Paolo Mori² and Carlo Taticchi^{1,*}

¹ *Università degli Studi di Perugia*

² *Istituto di Informatica e Telematica - Consiglio Nazionale delle Ricerche*

³ *Univesrità di Camerino*

Abstract

Certification of product origin and supervision of supply chains are fundamental activities in today's market scenario. Hence, it is of the highest interest to develop platforms that allow domain experts to quickly and easily build supply chain management systems allowing them to trace their production/transformation processes. This paper presents how the *-chain framework can solve this problem. In particular, we used the model and the graphical language defined by our framework to represent an olive oil supply chain, and the suite of tools we develop within such a framework to generate the related blockchain based traceability system, i.e., to automatically generate a set of solidity smart contracts implementing the system and two web interfaces to interact with them (one for supply chain administrators, the other for the actors of the production/transformation process), starting from the graphical representation.

Keywords

Supply Chain, Blockchain, Distributed Ledger Technology, Domain Specific Graphical Language, Smart Contracts, Automatic Smart Contract Generation.

1. Introduction

Supply chains are network of organizations, involved in the different processes and activities, that produce value in the form of products and services for the final buyer [6]. Depending on the specific scenario (e.g., product processing, service provisioning, warehousing) different types of supply chains can be considered. For instance, a production supply chain represents the flow of goods from the raw material to the final product. Supply chain management is therefore a crucial process to optimise the production cycle and lower the related costs. Blockchain technologies have been proven effective in developing solutions for the implementation of management systems for supply chains. Several works are currently available in this regard (like [1, 4, 5]), but these solutions are too specific for the field they were designed for. Consequently, they are not general enough to be used in alternative scenarios without major changes and adjustments, for which a relevant knowledge of the blockchain technology and expertise in smart contracts development is required. In this regard, we introduced the *-chain platform [3, 2], a platform to

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

✉ stefano.bistarelli@unipg.it (S. Bistarelli); francesco.faloci@unicam.it (F. Faloci); paolo.mori@iit.cnr.it (P. Mori); carlo.taticchi@unipg.it (C. Taticchi)

ORCID 0000-0001-7411-9678 (S. Bistarelli); 0000-0002-6413-6834 (F. Faloci); 0000-0002-6618-0388 (P. Mori); 0000-0003-1260-4672 (C. Taticchi)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

simplify the implementation of blockchain based supply chain management systems (SCMSs). Our platform provides a powerful Domain-Specific Graphical Language to represent supply chains, an easy-to-use graphical interface for authoring such representations, and a suite of tools for translating graphical models into the smart contracts building up the SCMS.

Our platform serves two main purposes: on the one hand, it provides a general solution for implementing supply chain management systems; on the other hand, it allows the supply chain administrators to implement applications distributed via blockchain technologies. In this paper, we demonstrate the capabilities of our platform, using the olive oil supply chain as a study case. In particular, we use *-chain to model all the steps required to produce marketable Olive Oil, starting from the harvesting phase up to bottling.

The rest of this paper is organized as follows: in Section 2 we introduce fundamental notions of blockchain and smart contracts; Section 3 describes *-chain functionalities, together with insights on its implementation; Section 4 summarises the Olive Oil supply chain, highlighting the main stages that the product goes through; in Section 5 we show how the considered supply chain can be modeled through our tool; Section 6, finally, summarises and concludes the paper, also pointing out new directions for future work.

2. Background

Distributed Ledger Technology (DLT) refers to systems and protocols that allow simultaneous access, validation, and updating with immutable data across a network. In simple words, the DLT is all about the idea of a "decentralized" network against the conventional monolithic centralized mechanism. Blockchain Technology (BT) is a special case of DLT, focusing on industries and financial sectors. The BT offers great potential to foster various sectors with its unique combination of characteristics as decentralization, immutability, and transparency. So far, the most prominent attention the technology received was through news from industry and media about the development of cryptocurrencies (such as Bitcoin¹, and Monero²), which all are having remarkable capitalization. BT, however, is not limited to cryptocurrencies; there are already existing blockchain based applications in industry and the public sector. Also, BT can have applications on non-financial sector, such as traceability problems and workflow organization. A smart contract is a self-executing contract (script) with the terms of the agreement between two actors, generally a buyer and a seller, directly written into lines of code. The code and the agreements contained in the script exist across a distributed decentralized blockchain system. One of the most popular coding languages for describing smart contracts is Solidity³, widely used for Ethereum⁴ systems.

¹Bitcoin Project: <https://bitcoin.org>

²Monero project: <https://www.getmonero.org>

³Solidity white paper: <https://docs.soliditylang.org/en/v0.8.6/>

⁴Ethereum project: <https://ethereum.org/en/>

3. *-chain framework

The *-chain framework [2] consists of a set of tools designed to aid the development of blockchain-based SCMSs. In particular, the end-user is provided with a web interface built upon a Domain-Specific Graphical Language (DSGL) that allows for tracing supply chains on the blockchain [3]. The framework translates the representation specified by the user into a set of smart contracts that are then used to implement an SCMS within the blockchain. The main purpose of *-chain is to enable experts in the context of a specific supply chain process (such as olive oil production) to contribute in the definition of SCMSs that can be distributed via blockchain, even with little or no knowledge of DLT. Therefore, our framework decouples the distinct tasks of supply chain process design and smart contract programming, which can be assigned to different professionals in the two respective fields.

The web interface provided by the *-chain framework integrates a graphical editor that can be used by supply chain domain experts to design SCMSs through a dedicated DSGL, equipped in turn with primitives that allow representing the most common types of supply chains. The editor also produces a textual, JSON-formatted, representation of the specified supply chain model, which is used to generate the smart contracts in the final SCMS. Those smart contracts corresponds to asset types belonging to a given supply chain and are endowed with functions that trace the execution of operations supported by the various assets.

4. Olive oil supply chain

The olive oil production process requires a series of steps that we can summarise in the following main phases:

- agricultural maintenance
- preparation of the olives
- crushing, kneading
- extraction
- separation
- conservation

During each of those steps, various parameters are monitored to guarantee the quality of the final product. Olive oil is, indeed, subjected to quality controls that determine its commercial classification⁵. For example, the temperature reached by the olive paste and the degree of oxidation of the oil can alter the final result, both in terms of taste/odour and chemical properties. Exceeding the imposed thresholds can cause, in addition to unpleasant smells and tastes, also harmful effects on the human body. In this section, we illustrate the whole olive oil supply chain, also providing, for demonstration purposes, insights into the agricultural maintenance phase.

The **agricultural maintenance** (for which we show a flow chart in Figure 1) constitutes the first phase of the olive oil supply chain and has a major impact on the organoleptic characteristics

⁵Olive oil classification follows the EEC regulation 2568/91 and subsequent amendments (656/95 and 2472/97), and the commercial standard of the International Olive Committee (Norma COI, 1998).

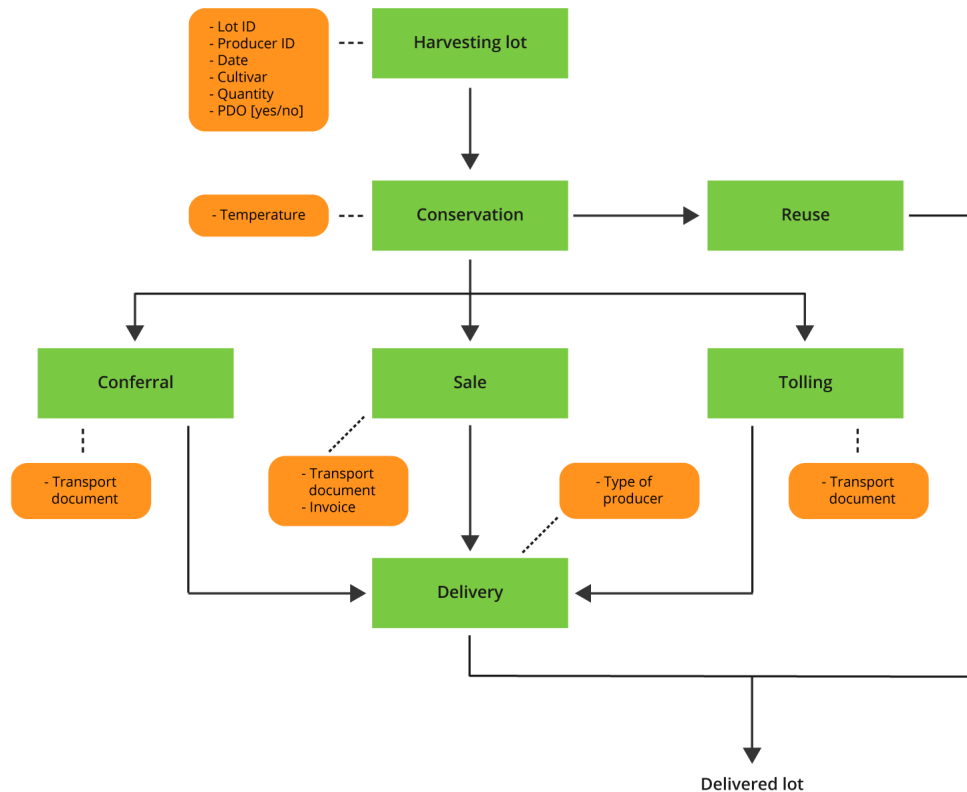


Figure 1: Flowchart fragment of the olive oil supply chain showing the stages of agricultural maintenance (green rectangles) along with the properties to be recorded (in orange with rounded edges).

of the final product. In this phase, olives are harvested, transported and then either sold or used in the following phases. Before the **processing** phase, olives undergo a series of operations, such as, for example, weighing, husking (elimination of foreign material), selection (division according to healthiness and size) and washing. Between weighing and husking there can also be short storage of the olives (maximum 24 hours) following the indications given for conservation in the agricultural phase. The weighing is carried out in the oil mill at the time of delivery. As for washing, this is done either by immersion or with special washing machines that maintain a forced movement of the water. In the **crushing** phase, the olive cell wall is broken in order to release juices. The crusher (with discs, hammers or knives) is mostly used in continuous cycle systems because it better responds to automation needs. The loading is carried out mechanically and the unloading takes place from the bottom, again mechanically, with the pouring of the oil paste into the kneading machines. The processing can take place in a very short time, and with a minimum footprint. **Kneading** is the phase in which oil drops aggregate and grow in volume. During this process, the olive paste coming from the crushing is slowly stirred to ease the separation of the oily component from the aqueous one. The temperature is an important parameter to monitor, as the pasta should reach a maximum of around 25 °C. Another discriminating factor in the final result is the kind of used machinery, which varies

from traditional millstones to modern malaxers. The **extraction** process consists in separating oil, water and solid parts in the kneaded paste. The factors that most affect the final result during this phase are again the temperature and the time needed to conclude the operation. Moreover, three main types of extraction technology can be used, each deriving from a different physical principle: pressure, percolation and centrifugation. The oil obtained after the extraction phase often contains a minimum percentage of water, which undergoes a further **separation** process, which can be carried out either through decantation or centrifugation. The final product will have a different yield depending on the methodology used. At this stage, the oil can already be consumed and it is stored in containers where residues settle on the bottom to make the product clearer. If the oil is intended for marketing, it is subjected to **filtration** and **bottling**. Filtration removes solid and colloidal impurities from the oil and can be performed by using different techniques. For bottling, hygiene and storage regulations must be taken into account.

To better demonstrate the functioning of the methodology we propose, we describe all the mandatory and optional steps involved in the **agricultural maintenance** phase. As we can see from Figure 1, this phase begins with the harvesting of olives from trees and the creation of the corresponding lot. The information to monitor regards which cultivars are grown, as well as the harvesting date and the quantity of olives in the lot. Furthermore, it is specified whether the lot contains a PDO (Protected Designation of Origin) production, together with the owner's data. After the harvest, the olives can be stored in cool, well-ventilated rooms for a period of time that does not exceed 2 or 3 days. The temperature of the room must be controlled since it can compromise the quality of the product. The olives, then, can be either directly reused by the same owner or transferred to third parties. In the latter case, there are three possible options: sale, tolling and conferral. For tolling and conferral a transport document is mandatory, while in the case of a sale, such a document can be replaced by a regular invoice. Finally, when the olives are delivered, we are interested in the type of producer to which the asset is transferred.

5. Supply chain model translation

In order to translate the control process of the Olive Oil supply chain, we must first adapt the various phases of the flow chart described in Section 4. The first step is to translate the components of each phase into a logic block paired with the DSGL's blocks of the *-chain framework. We have to identify the relevant objects of the workflow with respect to the focus of the analysis: the aim of this analysis is to supervise the Olive Oil, tracing the various production processes in order to accredit the origin of the goods. Each element we want to track is identified as an asset, paired with a blue rectangle shape. Each feature of this asset represents a relevant detail for the supply chain representation meaning. These features are paired with small grey circles attached to the blue shape of the asset itself: each circle is a property of the asset. Lastly, each activity applied to an asset is represented as operation: these operations are paired with a pointed arrow that links an asset to another object on the schema; the shape of the arrow defines the type of the operation.

Figure 1 shows a subsection of the workflow of olive oil supply chain. Figure 2 represents the translated schema: this schema is drawn with the *-chain framework, using one of the main tools of the platform: the design interface.

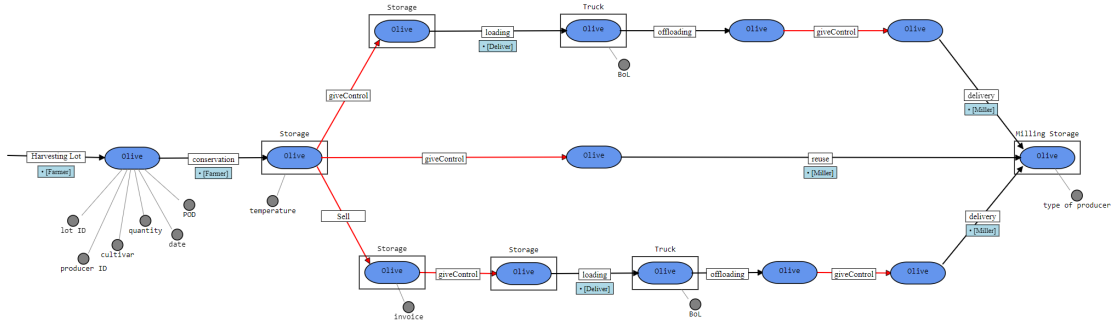


Figure 2: Representation of Olive oil supply chain with the *-chain framework.

5.1. Supply chain model representation

The first asset –the blue icon in the leftmost side of the schema– is the “*Olive*”, which is a countable and consumable asset; this asset does not have any incoming arrows from another asset: this could mean that: *i*) this is the point of origin of any asset *Olive*, or *ii*) the production of this asset is not tracked using this supply chain schema. In both cases, the asset *Olive* is generated at this point of the schema, using an *asset_create()* operation, labelled as “Harvesting Lot”. Under this create operation the label “*Farmer*” is specified, which means that the creation of an *Olive* asset can only be performed by a user paired with the “*Farmer*” role. The asset *Olive* is also defined with a set of properties, which are: “lot ID”, “producer ID”, “date”, “cultivar”, “PDO” and “quantity”. We can consider the *Olive* asset as the observation element of the supply chain. The second element of the schema is still a *Olive* asset, wrapped into a “*Storage*” container. This container should represent a room, a warehouse, or a silos, thus a generic location where the *Olive* asset is stored. The “*Storage*” container is a non-consumable container. The container *Storage* is defined with only a property: “temperature”. The two sides of these *Olive* assets are connected with a black arrow representing the operation “*conservation*”: *conservation* represents the conservation process of the **agricultural maintenance** phase. The “*conservation*” is an “*asset_pack()*” operation. This type of operation represents the process to transfer an asset into a specific item defined as container. Also under the “*conservation*” operation the label for the permission role is specified: even in this case, the operation can be performed only by a user paired with the “*Farmer*” role.

The next step is divided into three different paths drawn in the diagram. Each direction represents a different procedure: “Conferral”, “Tolling”, “Sale” and “Reuse”. The Conferral and Tolling procedures are identical under the procedural aspect, they differ only on their names. For the sake of simplicity these two procedures will be represented with the same path. Each of the different paths represents a plausible process choice. As it is described in the specifications of the **agricultural maintenance** phase, it is possible for an *Olive* asset to be sold to another user, transferred for processing to another user while maintaining the original owner, or simply reused for a subsequent phase.

The path that starts with a red pointed arrow oriented to the top right side is representing the Conferral and Tolling procedures. This path starts with a “*giveControl*” operation, establishing

the change of controller from a user with the “Farmer” role to a user with the “Deliver” role. The *giveControl* operation does not modify any other property of the asset *Olive*. through this operation, only the controller of the *Olive* asset is changed, meanwhile the owner is still a user with “Farmer” role. Following this path the *Olive* asset is loaded into a vehicle for the delivering; this procedure is represented as following: a “*Truck*” container represents the vehicle for the delivering procedure. The *Truck* is a non-consumable container with the bill of lading (“BoL”) descriptor as the only property. The “*loading*” operation is declared as “*asset_flow()*” -transferring an asset from a package to another-, loading the *Olive* into the *Truck*: this operation has a role constraint on user with “Deliver” role. Hence, at the destination, the asset *Olive* is unloaded from the *truck*; its controller is also changed from “Deliver” to “Miller”. These changes of the asset are represented by two arrows: the black arrow with the label “*offloading*” represents the asset being unloaded from the vehicle and placed in the “Miller”; the red arrow with the label “*giveControl*” represents the change of controller to a user with the “Miller” role. The user with the “Miller” role is the one that could load the asset *Olive* into the “*Milling Storage*”. This storage is a non-consumable container, and is the starting point for the **processing** phase. Last operation on this path is the “*delivery*” one. This operation is an “*asset_pack()*” that store the *Olive* asset into the *Milling Store*.

The path that starts with a red pointed arrow oriented to the bottom right side is representing the Sale procedure. This path starts with a “*sell*” operation, establishing the change of Owner from a user with the “Farmer” role to a user with the “Miller” role. The *sell* operation modifies the owner and the controller of the asset *Olive*. In the same way as the previously described path, the *Olive* asset is loaded into a vehicle for the delivering. After the *sell* operation, this scenario follows exactly the same procedure, ending with the deliver transaction. Therefore -for simplicity- we do not describe again the exact operations performed in this short step, since they are also performed in the same order. Also, the last operation on this path is -again- the “*delivery*”, following the same constraints.

The third path to be represented is that of reuse: in this step of the **agricultural maintenance** phase an *Olive* asset is stored by the same producer (the Owner). So there is no sale or transfer of control: the owner must also be a user with the “Miller” role. To perform operations on the *Milling Store* it is necessary for the user to have the “Miller” role.

5.2. Translation of the graphical representation

Once the design is complete, the *-chain framework translates the graphical model into smart contract skeletons: the translation tool collects the assets, operations and roles defined in the design interface. Each asset is represented by a different smart contract. Each smart contract contains:

- A data structure representing the history of all the assets of the same type.
- All explicit operations defined on the asset.
- The implicit operations of creation and destruction.
- The implicit operation of “*view()*” that provides the history of a given asset.

There are four main contracts in the smart contract skeleton: *contract_Olive*, *contract_Storage*, *contract_Truck* and *contract_Milling_Storage*. The main actor is *contract_Olive* which refers

to the asset Olive in the graphic representation. The other contracts are generated from the container elements, used for to store or to transfer the asset. The contract contract_Olive has all the declared property on the asset *Olive*, listed in lexicographical order. On this contract are also auto-generated and listed the functions: “conservation()”, “reuse()”, “loading()”, “offloading()”, “delivery()”, “sell()”, and “giveControl()”. In order to easily understand the structure of the auto-generated smart contract, a snippet code from the programming list of the solidity language is shown in Figure 3.

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >= 0.8.0;
3 import "@openzeppelin/contracts/access/AccessControl.sol";
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6 contract contract_Olive is ERC20, AccessControl {
7     bytes32 public constant OWNER_ROLE = keccak256("OWNER");
8     bytes32 public constant CONTROLLER_ROLE = keccak256("CONTROLLER");
9
10    enum asset_states {Initialized, conservation_ed, Sell_ed, giveControl_ed, reuse_ed, loading_ed, offloading_ed, delivery_ed, Destroyed}
11    struct {
12        //properties
13        address Owner;
14        address Controller;
15        string quantity;
16        string cultivar;
17        string producer_ID;
18        string lot_ID;
19        string POD;
20        string invoice;
21        string type_of_producer;
22        //states
23        asset_states state_of_Olive;
24    } asset_Olive_history;
25
26    struct {
27        asset_Gathering_history[] Olive_history;
28        uint256 ID;
29    } asset_Olive_struct;
30
31    mapping(uint => asset_Olive_struct[]) public assets_Olive;
32
33    function conservation(uint _ID) public {
34    }
35
36    function Sell(uint _ID) public {
37    }
38
39    function giveControl(uint _ID) public {
40    }
41
42    function reuse(uint _ID) public {
43    }
44
45    function loading(uint _ID) public {
46    }
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89

```

Figure 3: Snippet of Solidity code generated by *-chain framework, translating the graphical representation of Figure 2.

6. Conclusion and Future work

We have demonstrated how the *-chain framework can easily translate very complex supply chains: the DSGL developed for the platform manages to represent complex forms of process, providing easy-to-identify elements. The framework is also able to translate the supply chain schema into a solidity code listing, which is almost optimal to be deployed by a domain expert.

In future work we plan to better analyze the potential of the DSGL, comparing it with several other models and tools, aiming to underline differences or similarities. Also, we plan to develop a validation phase for the generated smart contract, to ensure a more clear and working output code. Furthermore, it would be interesting to analyse the robustness of the code validating the

model and the generated code through qualitative analysis. Our task is to refine the framework and make the graphical interface much easier to handle, especially for inexperienced managers who lack specific knowledge of the model. To further improve the usability of the framework, we plan to introduce the possibility of defining macro-functions, i.e., the composition of existing operations. The goal is to reduce procedural costs and earn an easier and clearer design. As a successive step, we plan to translate it into other languages for DLT, such as Chaincode⁶. Finally, we plan to investigate whether the adoption of different data structures to represent the asset history reduces the storage and execution costs of the proposed solution.

7. Acknowledgments

This work has been partially supported by:

- “GNCSINdAM”, codice CUP E55F22000270001
- “RACRA2018-2019” (research project 2018-2019)
- “DOPUP PSR REGIONE UMBRIA” 2014-2020
- “BLOCKCHAIN4FOODCHAIN” (research project 2020)

References

- [1] Rita Azzi, Rima Kilany Chamoun, and Maria Sokhn. The power of a blockchain-based supply chain. *Computers & Industrial Engineering*, 135:582–592, 2019.
- [2] Stefano Bistarelli, Francesco Faloci, and Paolo Mori. *.chain: automatic coding of smart contracts and user interfaces for supply chains. In *Third International Conference on Blockchain Computing and Applications, BCCA 2021, Tartu, Estonia, November 15-17, 2021*, pages 164–171. IEEE, 2021.
- [3] Stefano Bistarelli, Francesco Faloci, and Paolo Mori. Towards a graphical dsl for tracing supply chains on blockchain. In *(To appear in) Euro-Par 2021: Parallel Processing Workshops - Euro-Par 2021 International Workshops, Online Event, September 01-03, 2021*, Lecture Notes in Computer Science. Springer, 2021.
- [4] Khaled Salah, Nishara Nizamuddin, Raja Jayaraman, and Mohammad Omar. Blockchain-based soybean traceability in agricultural supply chain. *IEEE Access*, 7:73295–73305, 2019.
- [5] Julian Solarte-Rivera, Andrés Vidal-Zemanate, Carlos Cobos, José Alejandro Chamorro-Lopez, and Tomas Velasco. Document management system based on a private blockchain for the support of the judicial embargoes process in colombia. In Raimundas Matulevicius and Remco M. Dijkman, editors, *Advanced Information Systems Engineering Workshops - CAiSE 2018 International Workshops, Tallinn, Estonia, June 11-15, 2018, Proceedings*, volume 316 of *Lecture Notes in Business Information Processing*, pages 126–137. Springer, 2018.
- [6] Hartmut Stadtler and Christoph Kilger. *Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies*. Springer Publishing Company, Incorporated, 4th edition, 2008.

⁶<https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html>

AgriChain: Blockchain Syntactic and Semantic Validation for Reducing Information Asymmetry In Agri-Food

Pierluigi Gallo^{1,2,3,*}, Federico Daidone², Filippo Sgroi⁴ and Mirko Avantaggiato⁵

¹*Department of Engineering, University of Palermo, 90128 Palermo, Italy*

²*SEEDS s.r.l., academic spin-off of the Dept of Engineering at the University of Palermo, 90141 Palermo, Italy*

³*CNIT, Consorzio Nazionale Interuniversitario per le Telecomunicazioni, Italy*

⁴*Department of Agricultural and Forestry Sciences, University of Palermo, Palermo, 90128, Italy*

⁵*former employee of SEEDS s.r.l., 90141 Palermo, Italy*

Abstract

Information asymmetry affects the actors of all the segments of the agri-food supply chain and can arise many problems in the market along the production chain. Transactions of agri-food products are asymmetric because suppliers and buyers have different levels of knowledge on the provenance, value, quality, and freshness of food. Collusive relations among the agri-food chain actors, especially between controllers companies and controlled ones, can cause market failures as they influence customers' purchase decisions and severe health accidents when food safety is compromised. This paper proposes using blockchain technology to combat information asymmetry and collusive relations. In addition to transparency, cryptography and trusts, which are natively provided by the blockchain, our approach provides a twofold mechanism for validating crowd sensed data: first, a lightweight syntax validation is run before writing data in the blockchain (providing accountability also thanks to immutability); then, a dedicated smart contract runs semantic validation in scenarios with multiple data sources. This semantic validation may reveal collusive behaviours, downgrade colluding nodes and exclude or down-weight their data in future validations. The smart contract seals data that pass both validations adding metadata on data quality. Results prove the feasibility of our solution on Hyperledger Fabric under the assumption that the majority of nodes are honest. Experimental results demonstrate that our implementation of the twofold validation using smart contracts scales well with the dimension of the blockchain state. Our mechanism may greatly impact Product Certification and Designation of Origin as it may be applied to check specific requirements for raw materials, products, and production processes and protect from the collusion of controlling consortia and certification bodies.

Keywords

Agri-food, economy, blockchain, smart contract, information asymmetry, validation

1. Introduction

In the globalised society, and above all in the developed economies, the quality and safety of agri-food productions have received increasing attention from the consumer as a result of the evolution intervened in recent years, in terms of production and marketing of products of

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

✉ pierluigi.gallo@unipa.it (Pierluigi Gallo)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

vegetable and animal origin in a fresh and transformed state. This structural and functional evolution of the sector is mainly due to some aspects. In fact, in the agri-food sector, there are new products with differentiated and differentiable characteristics, including agricultural commodities, vegetable and animal productions, highly-processed and high-service products.

Moreover, in the agri-food system, there is a strong integration of the productive sector with the final consumer market in terms of information flows, knowledge of markets, and consumer needs and expectations. The economic and technical literature reports the growing importance of food quality and safety. These concepts are related to brands, information transparency, traceability of production and commercial chains, the fight against counterfeiting, and food fraud [24, 9]. However, in countries with high per capita income, the current health and nutritional needs, expressed by new lifestyles, determine a rethinking of production protocols that are increasingly attentive to the problems of resource sustainability and the protection of environmental ecosystems and biodiversity. Finally, the continuous evolution of consumers' tastes and preferences expressed over time by the variations in the demand should not be overlooked. To manage this new scenario, the public operator and institutional figures have provided regulations at national and international levels, disciplinary and production controls, certifications and quality protection, international agreements, and trading platforms. However, modern technologies require the adoption of systems that can support themselves by minimising human intervention in data collection and certification processes. In this context, information availability becomes fundamental for consumers because they quest for valuable information to perceive and evaluate the quality of products, recognise the added value, and increase willingness to buy or pay more. For all these reasons, this paper aims to analyse the role of information and novel ICT technologies in creating higher standards of quality and improving the functional efficiency of agri-food production markets by reducing information asymmetries on the demand side.

The contribution of this paper joins together economy and computer science; first, we explain the economic and technical implications of the information asymmetry in the agri-food market, then explore a possible solution to reduce such an asymmetry using blockchain technology. Blockchain has intrinsic traits such as transparency, trust, and traceability; these features help to solve the information asymmetry but, alone, they are not enough to guarantee the data accuracy and validity. Blockchain technology provides data immutability, accountability, and traceability, but it does not guarantee the data quality. In the agri-food sector, data quality is the cornerstone; therefore, a blockchain-based AgriChain platform for data quality is necessary. Using blockchain and smart contracts and applying a novel data validation methodology, we combat information asymmetry and its negative influence on the net value of investments, the ranking of agri-food companies and their capability to access credit for financing their activities. AgriChain uses multiple data sources, in which data are analysed by a set of smart contracts implementing a two-step validation logic (syntactic and semantic). The **syntax validation works before data are written on the blockchain**; it checks both the data and the user's identity and guarantees the accountability of the written information. Then, AgriChain smart contract applies a **semantic validation that works after data are written on the blockchain and 'seals' them**. This validation smart contract fights information asymmetry, providing transparency and data accuracy.

The distortions of information asymmetry in the food market are described in Section 3. The

actors and the roles in the agri-food supply chain are discussed in Section 4. Section 6 describes the AgriChain methodology for validating and assessing data quality. Experimental setup and results are presented in Section 7, then related works and conclusions are drawn, respectively in Sections 8 and 9.

2. Background

This section briefly introduces the key elements of the proposed architecture, namely ontology and blockchain.

2.1. Ontology

In computer science, ontology is a way to represent semantics (the meaning) through the definition of categories, properties and relationships expressed through description logic [12]. An ontological approach enables or simplifies deductive reasoning, classification, problem-solving, and the simplification of information exchange among systems. Deductive reasoning is entrusted to the semantic reasoner, software capable of carrying out reasoning on formalised knowledge bases. It is capable of elaborating the knowledge base according to some rules to validate and analyse the knowledge base itself and, therefore, infer logical consequences. In 1999, the W3C adopted the Resource Description Framework (RDF), which became standard in 2004. RDF is a data model used to represent ontologies; the atomic data entity is the semantic triple, a set of three entities: subject-predicate-object. Triples represent a statement on semantic data (e.g., “Alice is 30”, “Alice knows Bob”). SPARQL Protocol and RDF Query Language (SPARQL) is a SQL-like query language for receiving and manipulating RDF data. An implementation of SPARQL is included in Apache Jena, a Java framework for developing semantic web-oriented applications that include a SPARQL endpoint and supports a specific serialisation format named Turtle (Terse RDF Triple Language). RDF data validation is entrusted to Shapes Constraint Language (SHACL), which includes a list of constraints such as cardinality, range of values, etc. [7].

2.2. Blockchain

Blockchain is a distributed technology that allows for addition-only data storage. Each member of the distributed network (node) has its data replica on which it tracks every resource exchange (transaction) between participants. The transactions are grouped into blocks, linked together through a content hash, to form a chain. Members participate in the validation of transactions in order to add them to the blocks through a distributed consensus algorithm. There are several types of protocols, the most famous being Proof of Work (PoW), Proof of Stake (PoS), and Byzantine fault tolerance (BFT). Ethereum was the first blockchain platform that introduced smart contracts, small programs for validating transactions and performing the computation in a distributed way. Ethereum is a permissionless blockchain where anyone can participate in the network and participate in the consensus protocol.

Conversely, there are permissioned blockchains, such as Hyperledger Fabric (HLF), where participants need special permissions to be part of it. HLF is part of the broader Hyperledger

framework, which includes other distributed ledgers, libraries and tools, and the Linux Foundation supports it. Here the smart contracts are called chaincodes and enable to read (*query operation*) and write (*invoke operation*) the ledger. The ledger is included in a channel; nodes that participate in this channel can read, write and invoke smart contracts. An HLF instance can manage multiple channels and, therefore, multiple ledgers, defining different levels of scope for each node.

Since version 2.0, HLF supports chaincodes as an external service. In this case, the chain code management is independent of the node and allows us to define an endpoint where it is executed.¹ In this endpoint, we can also run more complex services, which the chaincode is capable of invoking, such as in [18] where external chaincodes are used to query external data sources. The call can be made in the single execution of the chaincode, or in case of longer processing times, the chaincode can exploit the oracle paradigm [5]. In this case, the chaincode emits an event that the service intercepts to start the computation of the request. When the service has finished the processing, it returns the output to the chaincode.

3. Information asymmetry and market distortion

From an economic point of view, it is well known the possibility to score the perceived quality of food products using a scale that spans from optimal to poor without interfering with its potential edibility. However, the hygienic and sanitary safety of the products to the final consumer markets is challenging to evaluate. Consumers have shown great interest in features defining food quality, thanks to an excellent spending capability and a more sensitive contest than in the past. Food quality is a multidimensional and dynamic concept [14]. Quality is “a complex value whose definition involves objective and subjective components. For this reason, quality is not a characteristic that can be immediately described or identified. However, it is an idea that each of us has concerning what we need to satisfy a specific need. The more the characteristics of a product correspond to the complex expectations we have concerning it, the more we will be inclined to consider its quality” [25]. It becomes essential to deepen the analysis on the perception of qualitative aspects, combining technical quality indicators with measures and models of customer satisfaction interpretation in the information economy’s theoretical context. Indeed, placing on the market certified quality products is reflected in an increase in production costs and therefore in prices. Certification requires an estimation of the economic value attributed to the quality perceived by the customers and the evaluation of the premium price concerning the different and greater willingness to pay.

Information is an element that affects the functioning mechanisms of the markets, providing a twofold perspective. On the one hand, the “control” and the “management” of the information asymmetry between supply and demand, through the policy of trademarks, certifications, and labelling of agri-food productions. On the other hand, national and international public and private organisations and institutions preside over voluntary standardisation and establish rules and procedures for controlling market transaction costs. They check company and collective brands as precise quality signals, signals of value and contribute to strengthening

¹Available at https://hyperledger-fabric.readthedocs.io/en/release-2.4/cc_service.html

the necessary operating conditions for the exchange, contributing to the reduction of the information asymmetry typical of imperfect markets [1, 20].

The quality of food production and the economic efficiency of the markets are closely connected and correlated to the growing role of information. This type of situation does not always safeguard the security and correctness of the information and the ability to choose given to informed consumers. From the point of view of the economic production efficiency of the markets, these elements contribute to creating a sort of functional distortions of the agri-food markets that can prevent their correct functioning under the profile of economic theory. These specific conditions seem to simultaneously produce disadvantages for producers and consumers in terms of the natural relationship between supply and demand, oriented to the balance of short and long term markets.

4. AgriChain actors and roles

The agri-food supply chain is composed of segments that cooperate to evolve the production process from field to fork. Information asymmetry typically manifests in the last segment of the supply chain affecting final customers but, in many cases, also influences other actors. The various segments concur to a holistic view of the good, including production and transformation processes. In case of partial or inaccurate information, two consecutive parts of the supply chain (e.g., production, transportation, transformation, stock) may experience information asymmetry too. For example, farmers know the history of the grain they grow - origin, timing, and treatments. This information may be hidden to the miller, whose knowledge is limited to storage in silos and the milling process. The same issue related to lack of knowledge occurs between miller and distributors and, more in general, in all the steps between different actors. The chain of value and responsibility that links those actors from farm to fork is affected by information asymmetry in all its links.

Farmers and industries need prompt and trusted information to make better decisions for growing or transforming agri-food products. The introduction of blockchain in the agri-food sector has represented a digital innovation aimed at increasing business income by reducing production inputs (and therefore of costs expressed at constant prices) and increasing the outputs (the quantity produced and therefore of revenues expressed at constant prices). Digital innovation is always aimed at increasing the company's competitiveness and technical and economic efficiency by optimising production factors and reducing variable costs. For example, accurate information on the state of plants brings to savings of water for irrigation, avoiding unnecessary wastes. The same happens for fertilisers and pesticides with knowledge on seasonal trends and infections. These decisions change the structure of production costs and positively affect the entrepreneur's net income.

The information asymmetry negatively influences production and marketing choices, and the potential problems along the supply chain may lead to market failure. An important issue is related to product certification about the designation of origin. Such certifications are characterised by strict requirements and are guaranteed by consortia and certification bodies. However, between the controlling and controlled entities may arise collusive relations, which are then difficult to discover and strongly affect the market. A recent example is given

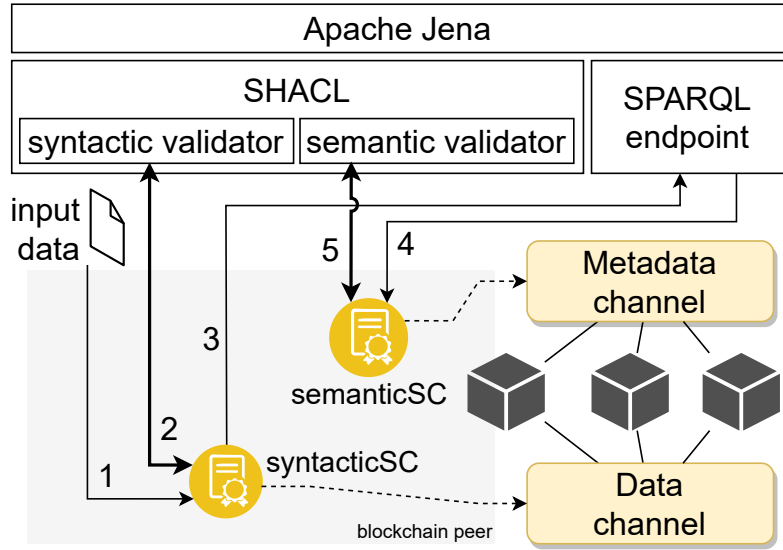


Figure 1: Blockchain node representation of our validation system. It includes a *SPARQL endpoint*, a *syntactic validator*, and a *semantic validator*

the production of ham under the Protected Designations of Origin (PDOs) “San Daniele” and “Parma”, which require the use of a specific breed of pigs. However, a collusive system within the protection consortium eluded controls on the seed of the pigs and, in contrast with the production disciplinary, put on the shelves products whose PDO was not valid. The effects of information asymmetry apply both to product quality and health, as in the cases such as pistachio, whose origin has implications in terms of aflatoxin and ochratoxin and may cause risks for consumers’ health [21]. This example shows that information asymmetry may have different facets. The consumer needs to know a product’s provenance, but this information is not sufficient if it is not linked to the risks of products from a specific area.

5. Validation Architecture

To combat the information asymmetry, we provide AgriChain, a blockchain-based platform for semantic and syntactic validation that executes external smart contracts on HLF (see, Section 2.2). In this way, within a blockchain node, we can run complex services such as *Apache Jena*, a free and open-source Java framework for building semantic applications [3], otherwise impossible to be implemented as legacy smart contracts. It includes a *SPARQL endpoint*, i.e. *Fuseki*, and a *syntactic and semantic validator*, i.e. *SHACL*. As shown in Figure 1, each node runs the two smart contracts in yellow that interface with the services mentioned above. We use HLF channels to separate the essential information from metadata and facilitate their operations. Semantic validation works on datasets rather than on a single transaction. The traceability information is stored in the *data channel*, while the *metadata channel* is used to store useful elements for the validation operations.

5.1. Syntactic validation

The syntactic validation takes place before storing the information on the blockchain, implementing filtering on the single input data. The smart contract *syntacticSC* (see, 1 in Figure 1) receives as input the data and performs a signature validation. This step is shown in Algorithm 1, where *syntacticSC* takes *dataIn* as input parameter and passes it to *signatureValidation()* function (see, Line 2). If that check is successful, we continue with syntactic validation, calling *syntacticValidation()* function (see, Line 3), as described in Section 6, to invoke the *syntactic validator* service (see, 2 in Figure 1). This validation has to be customised as needed and depends on the context of the application, for example, to verify that a “*weight*” field has a numeric value expressed in *kg*. When the validation is successful, we map *dataIn* into a *dataOut* format (see, Line 4) valid to be loaded on *SPARQL endpoint* (see, 3 in Figure 1). We assume that the reference ontology is preliminary written on the blockchain and imported into the *SPARQL endpoint* before starting the data collection process. Writing the ontology on the blockchain guarantees interoperability and transparency in the definitions of products and links between them. Finally, we also store *dataOut* on data channel (see, Lines 5 and 6).

Algorithm 1 syntacticSC

Require: *dataIn* as input data

```
1: procedure SYNTACTICSC(dataIn)
2:   if SIGNATUREVALIDATION(dataIn) then
3:     if SYNTACTICVALIDATION(dataIn) then
4:       dataOut ← MAPPING(dataIn)
5:       PUTSPARQL(dataOut)
6:       PUTBC(dataOut)
7:     end if
8:   end if
9: end procedure
```

5.1.1. Semantic validation

The semantic data validation process uses SHACL shapes, deriving from the ontology.² We assume that they are already present on the blockchain and used by the smart contract *semanticSC*. The *semanticSC*, as shown in Algorithm 2, receives as input the parameters *query*, that is, the SPARQL query which determines the subject of the validation, and *idShacl*, the identifier of a shape stored on the blockchain used in the validation. When this smart contract is invoked, it retrieves the *dataset* from the *SPARQL endpoint* (see, 4 in Figure 1), using *getSPARQL()* function with *query* parameter (see, Line 2). Similarly, we retrieve the *shacl* shape from blockchain with *getBC()* function (see, Line 3). Then we forward *dataset* and *shacl* shape to *SHACL validator* (see, 5 in Figure 1). Here, the *semanticValidator()* function calls the *semantic validator* service (see, Line 4) which performs a semantic validation and gives back the *result*. Now, *result*, along with

²We generated the SHACL shapes using the Astrea tool, <https://astrea.linkeddata.es/>. The shapes have been tuned, and we added missing validation elements from the ontology, such as the cardinality range.

dataset, are examined by a *calculateScore()* function (see, Line 5) which scores the validation performed. The single application defines the calculation of the score and its metric; for example, the closer the harvesting coordinates of different olives are, the more accurate the result that the crop belongs to an exact agricultural land. At the end, *query*, *idShacl*, *result*, and *score* are written on metadata channel as proof via *putBC()* function (see, Line 6).

Algorithm 2 semanticSC

Require: *query* as input query for SHACL validation

Require: *idShacl* as input id of SHACL shape

- 1: **procedure** SEMANTICSC(*query*, *idShacl*)
 - 2: *dataset* ← GETSPARQL(*query*)
 - 3: *shacl* ← GETBC(*idShacl*)
 - 4: *result* ← SEMANTICVALIDATOR(*dataset*,*shacl*)
 - 5: *score* ← CALCULATESCORE(*dataset*,*result*)
 - 6: PUTBC(*query*, *idShacl*, *result*, *score*)
 - 7: **end procedure**
-

The *reasonerSC* interfaces the blockchain with the reasoning service. When this smart contract is invoked, it queries the *SPARQL endpoint* (see 3 in Figure 1) to obtain the dataset to forward to the reasoner, indicated with 4). When the reasoner finishes its processing (see 5), the smart contract stores the result on the blockchain. If the result leads to new inferred triples from the initial dataset, the new data is updated in the *SPARQL endpoint* invoking *syntacticSC*. In such a case, the initial data are stored on the data channel, and the inferred information goes on the metadata channel.

6. AgriChain validation methodology

The agri-food sector includes multiple supply chains for the different agricultural products: tomatoes, wine, dairy, olive oil, etc. These supply chains involve many actors with different roles, and in most cases, they hold contrasting interests. Agricultural entrepreneurs, transformation industries, transport, logistics, and great and small distribution are exemplary actors that appear in many agri-food supply chains. However, any chain has its peculiar actors with specific needs and roles. For example, in the simplified model of the olive oil supply chain shown in Figure 2, there are farmers, olive growers' cooperatives, warehouses, shops, and customers as the main actors. These actors typically provide data through human operators, which are not trusted by default. To solve the problem of mistrusted operators, the authors propose to use IoT devices. However, this strategy shifts the point of trust from humans to IoT devices. IoT sensors are owned and maintained by those actors indicated above and can be maliciously manipulated according to their specific interests. To guarantee data quality, AgriChain leverages the double validation indicated above, invoking dedicated smart contracts.

The input syntactic validation, performed by smart contract *syntacticSC* (see, Pseudocode 1), checking that the transaction contains specific fields, as exemplary shown in Listing 1, including the actor's signature. The smart contract checks multiple signatures if multiple actors are

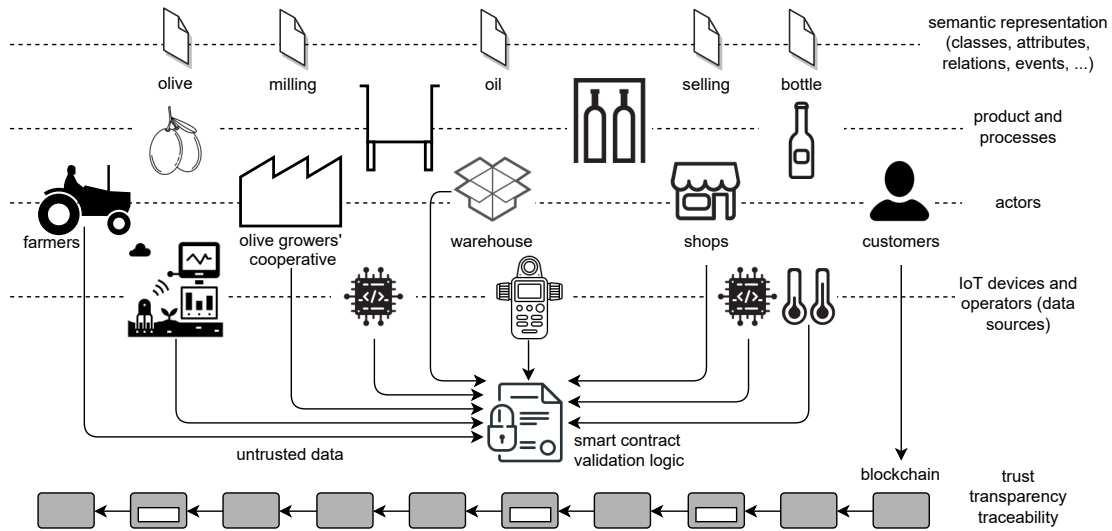


Figure 2: Data sources for AgriChain and smart contracts for data validation. Syntax-validated transactions are in gray, those semantically validated are in white.

involved in the transaction. This validation is performed on a transaction *before* being written on the blockchain. This preliminary validation guarantees accountability because each piece of data is linked to an accountable entity, but still, it does not protect from the ‘garbage in, garbage out’ problem. In other words, this lightweight syntax validation checks the identity of the data provider, the timestamp, and other metadata without guaranteeing ‘semantic’ validity.

```

{
  "actor": {
    "signature": "ebf3d6a0e54d249ff..." },
  "res_details": {
    "res_name": "olives01@field01",
    "hasGeoTag": true,
    "hasWeight": true },
  "data": {
    "lat": 38.120240,
    "lon": 13.357388,
    "kg": 10 },
  "ts": "2020-05-30T16:06:44+01:00"
}

```

Listing 1: Syntactic validation - Fields extracted from the transaction.

The second check involves both syntactic and semantic validation; in what follows, we stress the semantics aspects. Here, the smart contract *semanticSC* (see, Pseudocode 2) takes care of the validation on a more extensive set of data that, grouped, have a special meaning; the validation logic depends on the specific supply chain and the meaning of data, in our experiments we focused on the geographical origin of the olive oil product. Unlike the typical blockchain validation, our semantic validation is performed *after* the data is written on the blockchain, it is

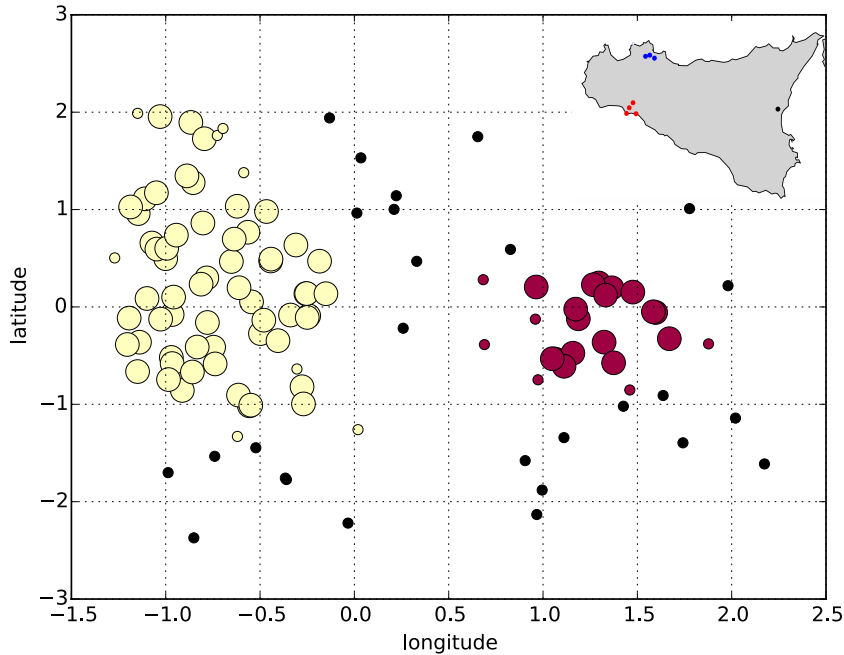


Figure 3: Exemplary semantic validation for geo coordinates of extra-virgin olive oil origin provided by 120 actors. The majority cluster (67 dots) is in yellow, noisy measures in black (30 elements), colluding nodes (23) in red.

triggered by new data arrivals that are semantically linked to the previous ones. For example, the geographic coordinates provided by several harvesting operators through their smartphones and IoT devices with GPS receivers are in Listing 1, providing the location of the product and farm *field01*. As shown in Figure 3, the syntax validation smart contract uses clustering to estimate the position (the mean of the majority cluster) from malicious and colluding nodes (in red).

6.1. Costs and benefits of the proposed solution

When an agri-food related business chooses to use blockchain technology to implement its food supply chain in some or all aspects, it is choosing to undergo some change. Change is not always good for business, so why should a business decide to switch to a blockchain-based solution? Because using blockchain expresses the company care about transparency, thus inspiring old customers to possibly buy more products and/or new ones to switch from another brand to this one. Of course, every kind of IT infrastructure comes with costs of installation and maintainability. We propose those costs to be proportionally assigned to the n involved actors. This solution could be thought of as a blockchain-based pay-per-use like a subscription system.

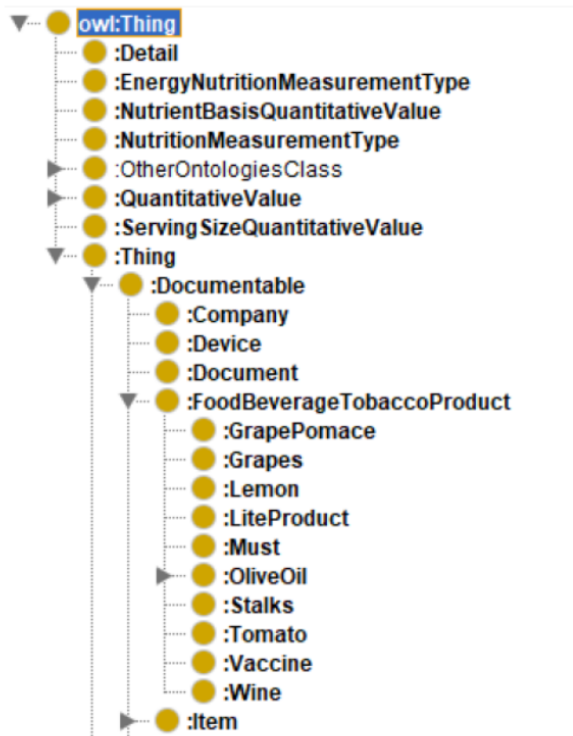


Figure 4: Protégé class hierarchy overview for *sb:OliveOil*.

7. Experimental Setup and Results

Part of the platform presented in this paper was proposed within the *DEMETER*³ project, which leads the digital transformation of the European agri-food sector through the rapid adoption of advanced IoT technologies, data science and smart farming, ensuring its long-term viability and sustainability. Our blockchain currently runs within the *DEMETER* ecosystem, and the project partners can invoke its services. A fundamental part is the semantic model, used as a common language between different project entities. It is based on the GS1 vocabulary, extended, revised and refined to be able to describe an entire supply chain. We exemplarily show the olive oil supply chain (see, Figure 4), where we have extended the *gs1:FoodBeverageTobaccoProduct*⁴ class with *sb:OliveOil*⁵ to be able to map the entire process. In addition to the interoperability offered by the semantic model and its mappings with other ontologies, the platform offers APIs compliant with the OpenAPI standard. Seeing the generality of the platform, we, as a case study, have implemented the validation of olive harvesting in the olive oil supply chain. Within the *SHACL validator*, we have added a clustering algorithm, the *DBSCAN* [19], to calculate the

³Available at <https://h2020-demeter.eu/>

⁴Available at: <https://www.gs1.org/voc/FoodBeverageTobaccoProduct>

⁵Available at: <https://seedsbit.com/ontology/#OliveOil>

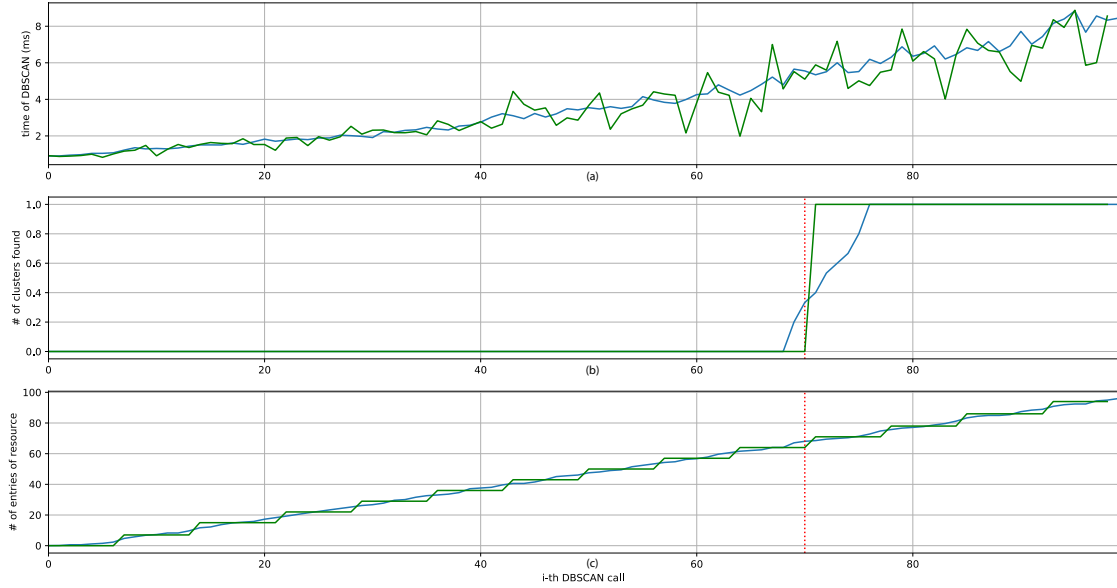


Figure 5: Time spent by the chaincode for clustering geographical points with DBSCAN for 100 consecutive invokes (a); Average number of points to create at least one cluster (in red) (b); Number of entries written on the blockchain (c). One typical run is depicted in green; the average value on 15 runs appears in blue. When a new pair of coordinates is added, the smart contract is triggered; the i th call works on a bigger state than the $i - 1$ th.

proximity of the harvested olives to the soil. Our blockchain platform of choice to illustrate our work is Hyperledger Fabric, although the SeedsBit platform uses multiple blockchain platforms, including MultiChain and Ethereum.

As introduced in Section 6, we used Hyperledger Fabric to implement our model partly and to give some experimental results in terms of performances. Our test network was composed of two Fabric organisations, having two peers each. Moreover, we used the RAFT algorithm [17], which is the default consensus protocol for Hyperledger Fabric. RAFT is a CFT (Crash-Fault Tolerant), but it can be easily substituted with a BFT (Byzantine Fault Tolerant) as Fabric has a modular approach to the consensus protocol [4].

Thus we had five nodes running for consensus purposes. The blockchain was deployed on a single host configuration on a machine with the following specs: Intel[®] Xeon[®] CPU E5-1660 v3 @ 3.00GHz with 32 gigabytes of RAM. Figure 5 shows, out of 100 consecutive invocations of the smart contract *semanticSC*, the time spent by the *DBSCAN* algorithm for clustering (see, Figure 5a), the number of clusters found (see, Figure 5b), and the number of entries used by *DBSCAN* (see, Figure 5c). At each invocation, we assume that the number of entries has increased by 1 unit, so *syntacticSC* has inserted a new entry into the blockchain. We can see how the analysis of 100 points, the most computationally expensive part, uses about 8 ms, with is compatible with the smart contract execution. The clusterisation of the terrain, with about 75 points, required 6 ms.

8. Related work

The problem of information asymmetry in food traceability has multiple facets that have been traditionally tackled singularly and using old paper documents and product specifications. Our approach towards information asymmetry is to improve transparency under multiple points of view: economy, blockchain technology, data quality.

From an economic point of view, it is well known the possibility to score the perceived quality of food products using a scale that spans from optimal to poor without interfering with its potential edibility. However, the hygienic and sanitary safety of the products to the final consumer markets are challenging to evaluate. Consumers have shown great interest in features defining food quality, thanks to a greater spending capability and a more sensitive contest than in the past. Food quality is a multidimensional and dynamic concept [14]. Quality is a complex feature made by objective and subjective components. For this reason, quality cannot be immediately described or identified, but it is a subjective idea that involves personal needs. The more the characteristics of a product match our expectations, the more we will be inclined to consider its quality [25]. It becomes important to deepen the analysis on the perception of qualitative aspects, combining technical quality indicators with measures and models of customer satisfaction interpretation in the information economy's theoretical context. Indeed, placing on the market certified quality products is reflected in an increase in production costs and therefore in prices. Certification requires an estimation of the economic value attributed to the quality perceived by the customers. This requires the evaluation of the premium price concerning the difference and greater willingness to pay. Information is an element that affects the functioning mechanisms of the markets, providing a twofold perspective. On the one hand, the "control" and the "management" of the information asymmetry between supply and demand, through the policy of trademarks, certifications and labelling of agri-food productions. On the other hand, national and international public and private organisations and institutions preside over voluntary standardisation and establishing rules and procedures for controlling market transaction costs. Company brands, collective brands, signals of quality and value work as media communication and contribute to strengthening the operating conditions necessary for the realisation of the economic exchange, contributing to the reduction of the information asymmetry typical of imperfect markets) [1, 2, 20].

From the point of view of the economic efficiency of the product markets, these elements contribute to creating a sort of functional distortions of the agri-food markets that can prevent their correct functioning under the profile of economic theory. These specific conditions seem to simultaneously produce disadvantages for producers and consumers in terms of the natural relationship between supply and demand, oriented to the balance of short and long term markets. In fact, in [22, 10, 23, 13] many different ways to leverage blockchain technology in this direction are illustrated. In [22] it is explained why a food traceability system based on RFID and blockchain would be ideal in China after many food safety accidents happened. These accidents were related to inadequate and primitive food supply chain management. In [10], the typical steps and places of a blockchain-based food traceability system are shown. The authors of [8] conclude their work stating that 'there are still few uses to support that some properties of blockchain implementation might be useful towards supply chain management'. In [13], it is reported how Walmart - one of the biggest American corporations in the hypermarket's

field - in collaborations with IBM, reduced the time needed to track the origins of mango “from seven days to 2.2 seconds”. These performances also show how blockchain is, without doubt, a solution to at least consider when talking about food safety and food supply management. The blockchain used in this pilot study was Hyperledger Fabric. Among others, we found the high customisation possibilities offered by Hyperledger Fabric and its growing community and scientific literature response and usage. We see in [11] that performance is not going to be an issue at least in terms of transactions/second (the authors state that - after heavy re-engineering - they reached 20000 transactions/second). On the other hand, in [16] we see possible problems in critical scenarios if the blockchain physical network undergoes latency. In addition to performance, the blockchain has been used for guaranteeing high-quality data [26, 15].

9. Conclusion and Future Work

Quality of food production and the economic efficiency of the markets are closely connected and correlated to the growing role of information. This type of situation does not always safeguard the security and correctness of the information and the ability to choose given to informed consumers. The central role of the agri-food sector requires quality of data because erroneous, malicious, and missing information affect the food supply chain in terms of quality and safety. This paper presented AgriChain as a mechanism for validating data syntactically before being included in the blockchain and semantically before being sealed. These two validations are executed through a distributed logic, implemented with one or more dedicated smart contracts. Typically the blockchain is the preferred technology when seeking trust, transparency and traceability among actors who do not trust each other or have contrasting interests. We demonstrated how AgriChain goes beyond this vision on data management, breaking the simplistic concept that data written on the blockchain are trustful because they have been validated in advance. Indeed, Agrichain performs only a lightweight validation before including the information into a block; this only guarantees accountability and syntax consistency. From the semantic point of view, the second validation guarantees first data cleaning second data quality assessment. AgriChain performs *data cleaning* applies clustering algorithms implemented as smart contracts on data collected through crowd-sensing. Standard data cleaning methods aim at detecting and removing repeated entries, detecting outliers, checking data volumes. In general, such methods do not deal with malicious data sources. Then, AgriChain smart contract checks accuracy, timeliness, completeness, uniqueness, and consistency [6] and provides KQIs, *Key Quality Indicators* which are added, as metadata, as a data seal on the blockchain. This paper presented a new methodology for using smart contracts to enforce a twofold validation and guaranteeing the quality of data for food traceability.

Acknowledgments

The authors would like to thank the SNAPP laboratory of Security, Network Applications and Positioning <http://www.unipa.it/SNAPPLab/> at the Department of Engineering of the University of Palermo, and SEEDS s.r.l. for experimenting on SeedsBit platform <https://seedsbit.com/>. This

work has been partially supported by the H2020 EU DEMETER project <https://h2020-demeter.eu/>.

References

- [1] George A. Akerlof. The market for "lemons": Quality uncertainty and the market mechanism. In *Decision Science*, pages 261–273. Elsevier, 2017.
- [2] Gervasio Antonelli. *Unione Europea, qualità agro-alimentare e commercio mondiale. Opportunità e minacce per i prodotti tipici delle Marche*. QuattroVenti, 2001.
- [3] Apache. A free and open source java framework for building semantic web and linked data applications, 2022.
- [4] Artem Barger, Yacov Manevich, Hagar Meir, and Yoav Tock. A byzantine fault-tolerant consensus library for hyperledger fabric, 2021.
- [5] Abdeljalil Beniiche. A study of blockchain oracles, 2020.
- [6] Hongju Cheng, Danyang Feng, Xiaobin Shi, and Chongcheng Chen. Data quality analysis and cleaning strategy for wireless sensor networks. *Eurasip Journal on Wireless Communications and Networking*, 2018(1), 2018.
- [7] Ben De Meester, Pieter Heyvaert, Dörthe Arndt, Anastasia Dimou, and Ruben Verborgh. Rdf graph validation using rule-based reasoning. *Semantic Web*, 12(1):117–142, 2021.
- [8] S. Matthew English and Ehsan Nezhadian. Application of Bitcoin Data-Structures & Design Principles to Supply Chain Management. *arXiv preprint arXiv:1703.04206*, 2017.
- [9] Huanhuan Feng, Xiang Wang, Yanqing Duan, Jian Zhang, and Xiaoshuan Zhang. Applying blockchain technology to improve agri-food traceability: A review of development methods, benefits and challenges. *Journal of cleaner production*, 260:121031, 2020.
- [10] Juan F. Galvez, J. C. Mejuto, and J. Simal-Gandara. Future challenges on the use of blockchain for food traceability analysis. *TrAC - Trends in Analytical Chemistry*, 107:222–232, 2018.
- [11] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. In *ICBC 2019 - IEEE International Conference on Blockchain and Cryptocurrency*, pages 455–463, 2019.
- [12] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.
- [13] Reshma Kamath. Food Traceability on Blockchain: Walmart's Pork and Mango Pilots with IBM. *The Journal of the British Blockchain Association*, 1(1):1–12, 2018.
- [14] Kelvin J. Lancaster. A New Approach to Consumer Theory. *Journal of Political Economy*, 74(2):132–157, 1966.
- [15] Danwei Liang, Jian An, Jindong Cheng, He Yang, and Ruowei Gui. The quality control in crowdsensing based on twice consensus of blockchain. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, pages 630–635, 2018.
- [16] Thanh Son Lam Nguyen, Guillaume Jourjon, Maria Potop-Butucaru, and Kim Loan Thai. Impact of network delays on Hyperledger Fabric. In *INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019*, pages 222–227, 2019.

- [17] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference ATC 14*, pages 305–319, 2014.
- [18] Srinath Perera, Amer A Hijazi, Geeganage Thilini Weerasuriya, Samudaya Nanayakkara, and Muhandiramge Nimashi Navodana Rodrigo. Blockchain-based trusted property transactions in the built environment: Development of an incubation-ready prototype. *Buildings*, 11(11):560, 2021.
- [19] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.
- [20] Joseph E Stiglitz. The causes and consequences of the dependence of quality on price. *Journal of economic literature*, 25(1):1–48, 1987.
- [21] Seyedeh Faezeh Taghizadeh, Ramin Rezaee, Gholamhossein Davarynejad, Javad Asili, Seyed Hossein Nemati, Marina Goumenou, Ioannis Tsakiris, Aristides M Tsatsakis, Kobra Shirani, and Gholamreza Karimi. Risk assessment of exposure to aflatoxin b1 and ochratoxin a through consumption of different pistachio (*pistacia vera l.*) cultivars collected from four geographical regions of iran. *Environmental toxicology and pharmacology*, 61:61–66, 2018.
- [22] Feng Tian. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In *2016 13th International Conference on Service Systems and Service Management, ICSSSM 2016*, pages 1–6. IEEE, 2016.
- [23] Feng Tian. A supply chain traceability system for food safety based on HACCP, blockchain & Internet of things. In *14th International Conference on Services Systems and Services Management, ICSSSM 2017 - Proceedings*, pages 1–6. IEEE, 2017.
- [24] S. Vieri. Quality Products and Genetically Modified Organisms in Italy: Hazards and Possible Enhancements. *Journal of Nutritional Ecology and Food Research*, 1(1):68–77, 2013.
- [25] S. Vieri. *Conflitti di maniera e accordi di sostanza*, 2015.
- [26] Jingzhong Wang, Mengru Li, Yunhua He, Hong Li, Ke Xiao, and Chao Wang. A Blockchain Based Privacy-Preserving Incentive Mechanism in Crowdsensing Applications. *IEEE Access*, 6:17545–17556, 2018.

On-Chain Global Maintenance Services*

Alessandro Bellini³, Antonio Bonifacio³, Salvatore Esposito De Falco²,
Simone Naldini³, Francesco Pacileo², Diego Pennino¹, Maurizio Pizzonia¹,
Domenico Sardanelli², Andrea Vitaletti², Pietro Vito² and Marco Zecchini^{2,*}

¹Università degli Studi Roma Tre, Dipartimento di Ingegneria, Sezione Informatica e Automazione, Via della Vasca Navale 79, 00146 Rome, Italy;

²Sapienza Università di Roma, Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Via Ariosto 25, 00185 Rome, Italy;

³Mathema, Via Torricoda 29, 50142 Florence, Italy;

Abstract

Facility management deals with all activities that are not core business for a company and are consequently outsourced to specialized companies. Maintenance is a fundamental activity in facility management and it is often handled by *Global Maintenance Services* (GMS) where some maintenance activities are delegated by the company to service providers and are remunerated according to measurable results expressed as Key Performance Indicators. In this context, it would be desirable to have information systems trustable by all involved actors. In this paper, we discuss the design of a blockchain solution capable to support a GMS on-chain. We first introduce the GMS concept and how it is related to the Principal-Agent relationship, then we show a reference architecture to implement GMS on-chain. We discuss a use case of on-chain GMS in a hospital showing how smart contracts and oracles can be used in this context. We present the advantages of this approach and we discuss the open problems for realizing a proof-of-concept.

Keywords

Maintenance, Blockchain, Oracle, Smart Contract, Sensors

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

* This research was partially funded by Sapienza Ateneo Research grant “La disintermediazione della Pubblica Amministrazione: il ruolo della tecnologia blockchain e le sue implicazioni nei processi e nei ruoli della PA”. This research was partially funded by Italian Minister of Economic Development (MISE) grant “ReASSET: Piattaforma Maas (Maintenance as a Service) Blockchain-IoT per la gestione operativa in tempo reale degli asset tecnico-impiantistici”. This research was partially funded by POR FESR LAZIO 2014 – 2020, call for “Gruppi di ricerca 2020”. Det. n. G04052 of Apr. 4th, 2019, under the “LazioChain” project, CUP F85F21001550009 - POR project code A0375E0116.

*Corresponding author.

✉ salvatore.espositodefalco@uniroma1.it (S. E. D. Falco); francesco.pacileo@uniroma1.it (F. Pacileo); pennino@ing.uniroma3.it (D. Pennino); pizzonia@ing.uniroma3.it (M. Pizzonia); domenico.sardanelli@uniroma1.it (D. Sardanelli); vitaletti@diag.uniroma1.it (A. Vitaletti); pietro.vito@uniroma1.it (P. Vito); zecchini@diag.uniroma1.it (M. Zecchini)

ORCID 0000-0001-5339-4531 (D. Pennino); 0000-0001-8758-3437 (M. Pizzonia); 0000-0003-1074-5068 (A. Vitaletti); 0000-0002-2280-9543 (M. Zecchini)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

1. Introduction

Facility Management (FM) deals with managing the facilities, namely all assets, both tangible and intangible, that support a company's core business making the life of occupants of residential buildings, shops, offices or factories more pleasant and safe. Within FM, a Global Maintenance Service (GMS), is a form of outsourcing contract specifically related to maintenance and based on measurable results. Through a GMS contract, a *client*, or *principal*, entrusts a series of activities aimed at the maintenance of the facilities to a single *primary service provider*, or *agent*, for a well defined period of time. The following elements are relevant to this paper for a GMS contract.

- The contract is based on results. The remuneration is a function of a series of Key Performance Indicators (KPIs) through which it is possible to measure the quality, efficiency and effectiveness of the performed activities.
- There is a working group made up of representatives of the client and the primary service provider, whose function is to ensure the correct start and execution of the project, with particular regard to the implementation of integrated management tools.
- The primary service provider appoints a single manager, with respect to which the client can refer as the sole interlocutor and who has responsibility for the activity of all the personnel involved in the performance of the services covered by the contract. The primary service provider can delegate some activities to *secondary service providers*.

Figure 1 summarizes and clarifies the relations between the parties discussed so far.

Examples of the employment of GMS contract for the maintenance of facilities include the following.

Lighting. Energy supply and ordinary and extraordinary maintenance of related systems.

Real estate assets. Their ordinary and extraordinary maintenance, plant maintenance, cleaning and surveillance services.

Green. Paving, cleaning, cutting of the grass, refurbishment of green areas.

Heat. Ensure the heating and air conditioning system including the supply of fuel, gas and electricity.

GMS can be modelled as a relationship between a Principal (P) and an Agent (A) [4], where the principal appoints the agent to act on its behalf for the maintenance of its facilities. According to the GMS, the relationship is governed by a contract (C) based on results measurable by suitable KPIs.

Usually, the client pays the provider either on the basis of measurements declared by the provider or by performing measurements by themselves. In the first approach, the client must trust the provider. In the second approach, the costs of the client for autonomously performing the measurements might be too high with respect to the benefits of the outsourcing approach.

In this paper, we propose an architecture based on blockchain and IoT technologies to address this problem. We also provide details for a sample use case of this approach encompassing oracles to acquire measurements from IoT devices into the blockchain. Our sample use case

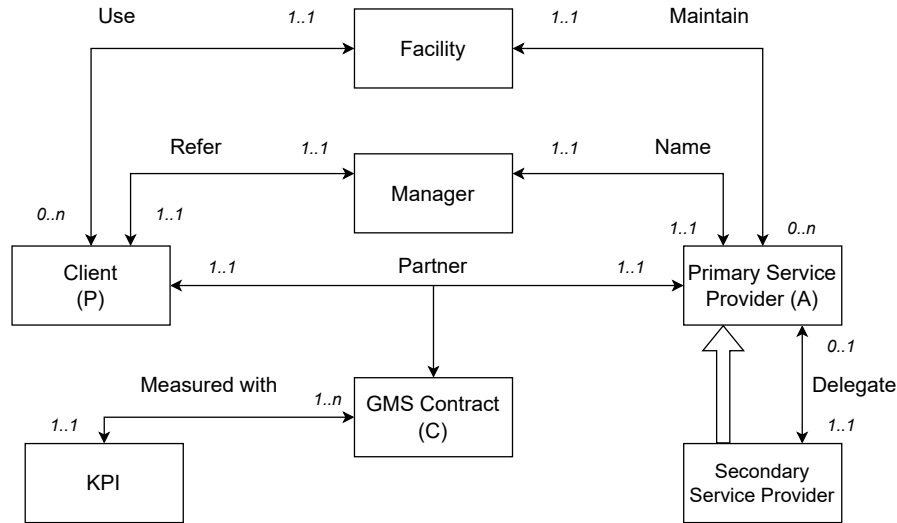


Figure 1: Diagram of the relations between parties

is taken from a real tender for heat maintenance related to an Italian hospital. We provide examples of smart contract code to realize that use case.

This paper is structured as follows. In Section 2, we provide background notions about blockchain technologies and how they are able to access off-chain data through oracles. In Section 3, we describe the architecture of a blockchain-based GMS and how it benefits from this technology. Finally, Section 4 draws the conclusions of the paper and provides some discussion about open problems.

2. Blockchain Background

A *blockchain* is a type of *Distributed Ledger Technology* (DLT) where transactions, new records to the ledger, are recorded according to an immutable order obtained by means of *cryptographic hash functions* that chain the blocks in which transactions are recorded. Unlike a centralized database, a blockchain is decentralized, namely there is no need for a central authority or intermediary for processing, validating, and/or authenticating transactions. A blockchain is typically managed by a set of autonomous *nodes* that collectively create a peer-to-peer (p2p) network adhering to a protocol for inter-node communication and validating new blocks. Nodes do not trust each other and malicious nodes are tolerated, within certain limits that depends on the consensus algorithm. The most common blockchains can be abstracted as a key-value database. For example, in a blockchain implementing a cryptocurrency, keys are *addresses* (or *accounts*), while values are the balances of their *wallets*. In this scenario, a transaction is an operation that transfer some cryptocurrency from a wallet to another. For efficiency reasons, transactions are not confirmed one-by-one but aggregated into *blocks*. Transactions are confirmed when a new block is *created* (or *mined*). The mining of a new block requires to verify that all transactions of the block, considered in the chosen order, comply to certain

consensus rules (that depends also on the application domain). When a new block is mined by a node of the network, all the other peers verify that it respects the consensus rules in a process called *validation*. The *consensus* (for more details, see [16, 17, 8]) is the decentralized process by which a block is finally stored in the ledger.

There are two types of blockchains that can be categorized according to the access in reading and writing to the content of the ledger and to the access in participating to the consensus. In public *blockchains*, everyone can read the content of the ledger and propose new transactions that, if successfully validated by the consensus, will be eventually stored in the ledger. On the contrary, in *private* blockchains, users are authenticated and access control allows or denies each user operation as occurs for access control of regular information systems. Similarly, in *permissionless* blockchain every user can participate in the consensus, while in *permissioned* one the participation in the consensus is allowed only to specific users.

While initially blockchain has been primarily conceived to implement cryptocurrency trading, it can now be adopted to realize general-purpose applications through the use of *smart contracts*. They consist of pieces of code that are executed as part of a transaction. In simple terms, in these cases, the blockchain implements a global decentralized virtual machine and smart contracts are the programs running on it.

Smart contracts can process only data that are stored in the blockchain. However, in the GMS use case that we consider in this paper, there is the need of accessing off-chain data. This is possible using an *Oracle* (for more details, see [6]). Oracles are components that allow a blockchain, or a smart contract, to get inputs from outside the blockchain through regular blockchain transactions. There are several oracle services providing APIs to allow smart contracts to access external data. Examples include Chainlink [1], Provable [3], BandChain [5], and Tellor [2].

3. GMS On-Chain

We have seen that GMS can be modeled as a P/A relationship (see Figure 1). In such a relationship, the agent acts on behalf of the principal and should not have a conflict of interest in carrying out its task. An agent may act in its own best interests and in a way that is contrary to the best interests of the principal, generating the so-called *P/A Problem*. This problem typically arises when P has into enough information to directly ensure that A is always acting in P's best interest.

The transparency, immutability, traceability and algorithmic governance offered by *Blockchain* technologies can contribute to mitigate the P/A Problem [10], reducing (or even eliminating) the asymmetry of information and thus facilitating the creation of a genuine net value.

The employment of the blockchain, allows us to envision new models of governance, where trust between the actors is substituted by A and P relying on the consensus within the P2P blockchain infrastructure, i.e. relying on a community rather than on the trust in individual actors. In this perspective, the natural different interests of P/A, at least economically-wise, as well as the participation of different providers competing in the market, are guarantees to the achievement of a real consensus among the parties, even in less open infrastructure such as the permissioned blockchains.

In particular, the Blockchain can provide:

- algorithmic governance autonomously managed by smart contracts capable to implement decentralized decision-making processes providing the highest guarantee of impartiality to all the involved stakeholders;
- a transparent and immutable bidding process to select the service providers (i.e. Agents);
- a minimization of the information coordination costs on a shared infrastructure, making the organization's data accessible to new customers and suppliers;
- a reduction of verification costs, namely costs involved in verifying the transactions between Principal and Agent.
- a reduction of intermediation costs, i.e. the costs due to the certification activities by a third party, external to the contractors.

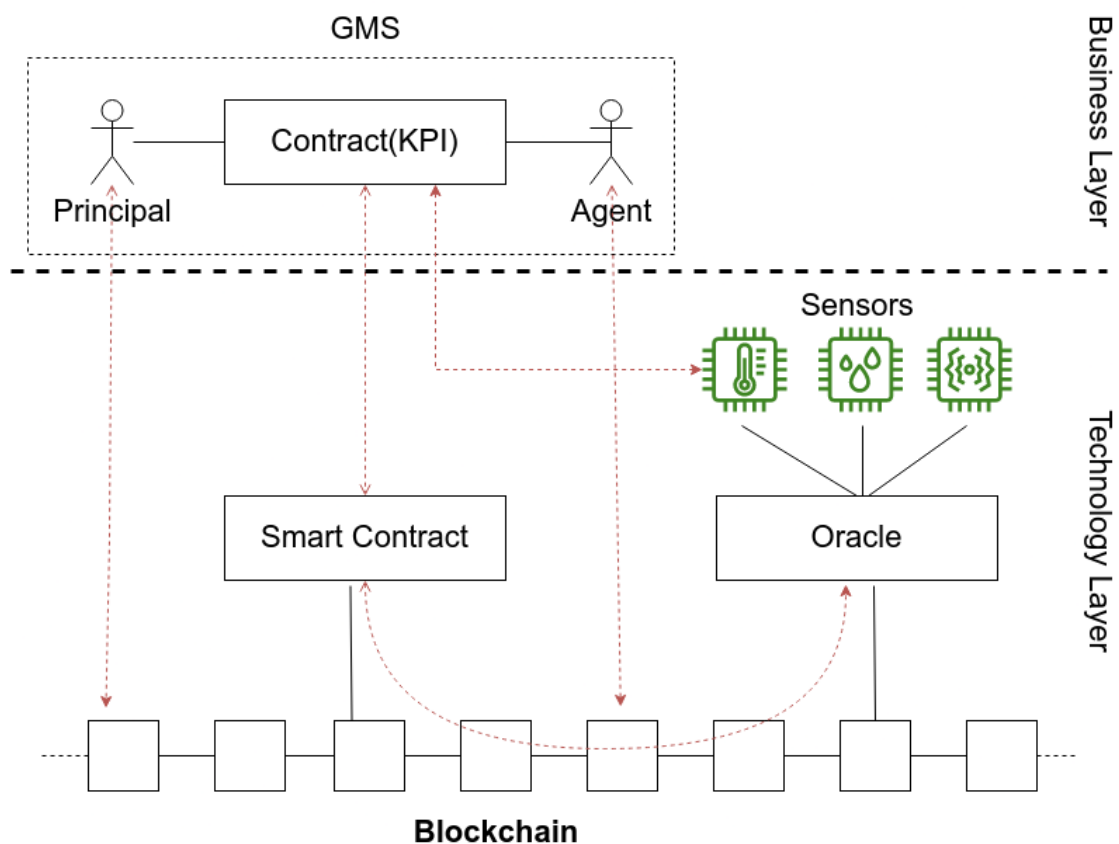


Figure 2: At the Business layer, the relationship between the Principal and Agent modeling the GMS, is regulated by a contract based on performance measures defined by suitable KPIs. The Business components are mapped into their technical counterparts in the Technical Layer. Principal and Agents are two entities on the Blockchain. The remuneration of the Agent (i.e. a transaction from the Principal to the Agent) is governed by a Smart Contract according to specific sensors' observations measuring the KPIs. To access sensors' data, the Smart Contract interacts with an Oracle.

3.1. Modelling the P/A Relationship On-Chain

The architecture of the on-chain GMS modeled as P/A relationship is represented in Figure 2. Here we assume that both the Principal and the Agent are entities on-chain identified by an address. Note that we currently assume the pseudoanonymity sufficient to carry on the economic transaction behind the GMS contract, however, while this is technically more convenient, we have not yet properly investigated all the complexity of managing identity on-chain [12] in particular when norms, laws, and regulations must be satisfied.

In the following, we will use Solidity code sketches to illustrate the structure and main components of the necessary smart contracts. Smart contracts of the following examples refer to the hospital heating use-case described in Section 3.2. The GMS contract is translated into a smart contract (see Listing 1). The function *payAgent*, at Line 23, performs the payment if the KPI are satisfied. In this case, this occurs when the level of CO emission measured by a sensor is below a given threshold (see line 27). To access the data of the IoT sensors [13], the smart contract interacts with an Oracle [7] as sketched in Listing 2. In this case, we use the Provable Thing Oracle [3], that provides access to off-chain data to a number of Blockchain Technologies, including Ethereum, EOS, R3 Corda and Hyperledger Fabric.

In some cases, primary service providers can take advantage of sub furniture provided by secondary service providers. Also in this case, a smart contract can be employed to manage this relationship as a P/A one as shown in Listing 3.

3.2. Use-Case: GMS for an Hospital

In this section, we show how the components of Figure 2 can be mapped to a real tender specifications document [11] which defines the modalities through which the Bianchi Melacrino Morelli Hospital in Reggio Calabria, Italy, intends to entrust the ordinary and extraordinary maintenance service of the buildings, the technical plants, and the furniture to a primary service provider for three years.

Art. 6 of [11], details the reference maintenance plan, and provides a number of sheets that list all the maintenance operations that the primary service provider has to perform. For the sake of simplicity, we simply consider the sheet in Table 1.

Maintenance guide N. 05		
Plant or Facility type	Operations performed by the maintenance service	Cyclicality
thermomechanical plants	Combustion control according to Legislative Decree 152/06	Annual

Table 1

Maintenance operations carried by the primary service provider

The sheet mandates that every year, the combustion of thermo-mechanical systems must be checked according to the D.L.gs 152/06 [14] regulation.

Principal and Agent The Principal is the hospital company. The Agent is the primary service provider taking care of the maintenance of the hospital facilities as foreseen in the GMS according to the specification provided [11] and possibly taking advantage of secondary service providers.

Measurable Contract Art. 286 of D.Lgs 152/06 [14] defines the threshold values and the measurement modalities to check the emissions. In the following, we summarize the most relevant elements for the considered use case.

- The atmospheric emissions of civil thermal plants with nominal thermal power above the threshold value must comply with the limit values set out in part III of Annex IX to part five of D.Lgs 152/06 (see Table 2).
- The emission values of the plants must be checked at least annually by the person in charge of the operation and maintenance of the plant during normal inspection and maintenance operations. The measured values, with the indication of the relative dates, of the measurement methods used and of the person who carried out the measurement, must be attached to the plant logbook.
- For the purposes of sampling, analysis and assessment of emissions from thermal plants referred to in paragraph 1, the methods provided for in part III of Annex IX (see Table 2) are applied.
- The installer verifies compliance with the emission limit values.

	Installed Electrical Rated Output (MW)			
	[1] > 0, 15% ≤ 3	> 3% ≤ 6	> 6% ≤ 20	> 20
Total dust	100mg/Nm ³	30mg/Nm ³	30mg/Nm ³	30mg/Nm ³
Total organic carbon (TOC)	n.a.	n.a.	30mg/Nm ³	20mg/Nm ³ 10mg/Nm ³
Carbon monoxide (CO)	350mg/Nm ³	300mg/Nm ³	250mg/Nm ³ 150mg/Nm ³	200 100mg/Nm ³
Nitrogen oxides (expressed in NO ₂)	500mg/Nm ³	500mg/Nm ³	400mg/Nm ³ 300mg/Nm ³	400mg/Nm ³ 200mg/Nm ³
Sulphur oxide (expressed in SO ₂)	200mg/Nm ³	200mg/Nm ³	200mg/Nm ³	200mg/Nm ³
[1] For plants with a rated thermal input equal to or greater than 0.0035 MW and no greater than 0.15 MW, it is applied an emission value for total dust of 200				

Table 2

An example of requirements and KPIs for a tender. Values shown in this table are taken from the Italian regulation [14] (see text), which applies to the the tender considered in Section 3.2.

According to the GMS, the smart contract will (a) verify the satisfaction of the requirements for the emissions according to the measured KPIs and the limit defined in [14] and (b) perform the payments to the service providers.

KPI and sensors The KPI to measure the satisfaction of the contract terms is clearly defined in Table 2. As an example, if we consider the Total Suspended Particles and a heating system with nominal installed power between 3 and 6 MW, the threshold is 30mg/Nm³.

The measurements of the satisfaction of the KPI are provided by the sensors of Table 2, namely Total Suspended Particles, Total Organic Carbon, Carbon Monoxide, Nitrogen Oxides, Sulfur Oxides. Sensors should sample the environment with accuracy and a sampling period defined by the regulation.

Smart Contract examples. Listing 1 illustrates how Principal and Agent can manage their collaboration with a smart contract. “PayCOContract” accesses external data through the contract oracle in Listing 2. Finally, “Subcontract” in Listing 3, demonstrates how a primary service provider can delegate a secondary service provider on a blockchain through a specific smart contract.

```

1  pragma solidity ^0.4.22;
2  import "../ExampleContract_CO_Oracle.sol";
3
4  contract PayCOContract {
5
6      ExampleContract_CO_ORACLE private OracleContract;
7      address public P;
8      address public A;
9      uint public CO_Threshold;
10     uint public lastPayment;
11     uint public payment;
12     uint public INTERVAL = 10;
13
14     constructor (address _OracleContract, address _P, address _A, uint _CO, uint _payment) {
15         OracleContract = ExampleContract_CO_ORACLE(_OracleContract);
16         P = _P;
17         A = _A;
18         CO_Threshold = _CO;
19         payment = _payment;
20         lastPayment = block.number;
21     }
22
23     function payAgent() payable {
24         require(block.number > lastPayment + INTERVAL);
25         require(msg.value == payment);
26         require(msg.sender == P);
27         if (stringToUint(OracleContract.CO) < CO_Threshold) revert();
28
29         A.transfer(msg.value);
30     }
31 }
32 }

```

Listing 1: A sketch of a smart contract to execute the payment from the Principal to Agent if specific conditions are met. For the sake of brevity “stringToUint” function is omitted but it casts a string into an unsigned integer in Solidity.

```

1  pragma solidity ^0.4.22;
2  import "github.com/provable-things/ethereum-api/provableAPI_0.4.25.sol";
3
4  contract ExampleContract_CO_ORACLE is usingProvable {
5
6      string public CO;
7      event LogConstructorInitiated(string nextStep);
8      event LogCOUpdated(string CO);
9      event LogNewProvableQuery(string description);
10 }

```

```

11 function ExampleContract() payable {
12     LogConstructorInitiated("Constructor was initiated. Call 'updateCO()' to send the
        Provable Query.");
13 }
14
15 function __callback(bytes32 myid, string result) {
16     if (msg.sender != provable_cbAddress()) revert();
17     CO = result;
18     LogCOUpdated(result);
19 }
20
21 function updateCO() payable {
22     if (provable_getPrice("URL") > this.balance) {
23         LogNewProvableQuery("Provable query was NOT sent, please add some ETH to cover
            for the query fee");
24     } else {
25         LogNewProvableQuery("Provable query was sent, standing by for the answer..");
26         provable_query("URL", "json(https://api.sensor.it).CO");
27     }
28 }
29 }

```

Listing 2: A sketch of a smart contract to get data from a CO sensor by Provable Things Oracles. Provable Things provide oracles for a number of Blockchain Technologies, including Ethereum, EOS, R3 Corda and Hyperledger Fabric

```

1 pragma solidity ^0.4.22;
2 import "./PayCOContract.sol";
3
4 contract Subcontractor is PayCOContract {
5     address public Secondary;
6     uint public SecondaryPayment;
7
8     constructor (address _OracleContract, address _P, address _A, uint _CO, uint _payment,
        address _S, uint _secondary_payment) PayCOContract(_OracleContract, _P, _A,_CO,
        _payment) {
9         Secondary = _S;
10        SecondaryPayment = _secondary_payment;
11    }
12
13    function paySecondary() public {
14        require(block.number > lastPayment + INTERVAL);
15        require(msg.value == SecondaryPayment);
16        require(msg.sender == A);
17        if (stringToUint(this.OracleContract.CO) < CO_Threshold) revert();
18
19        Secondary.transfer(msg.value);
20    }

```

Listing 3: A sketch of a smart contract to execute the payment from the Primary Service (A) to Secondary Service provider if specific conditions are met. Note that, since this is a special case of contract where the agent delegates to another entity the management of some facility, Subcontract inherits the main contract "PayCOContract"

4. Conclusions

In this paper, we discuss the design of a blockchain solution capable to support the Global Maintenance Service on-chain, the implementation of the Principal/Agent relationship and how modeling of the GMS on-chain provides several advantages. The transparency of Blockchain can eliminate the asymmetry of information and consequently, it reduces (or even eliminate) the P/A problem and allows a transparent and immutable bidding process for the selection of Agents. The algorithmic governance autonomously managed by smart contracts is capable to implement decentralized decision-making processes providing the highest guarantee of impartiality to all the involved stakeholders and the shared blockchain infrastructure allows us to minimize the information coordination, verification and intermediation costs.

All these arguments encourage us to proceed in this investigation, however, a number of relevant questions still need to be properly handled.

The selection of the most suitable blockchain technology is the first relevant issue. Public/permissionless blockchains provide the highest guarantees but could be difficult to be implemented in an industrial context where some information is necessarily sensitive and private. However, the natural different interests of Principal and Agents, at least economically-wise, as well as the participation of different providers competing in the market, are guarantees to the achievement of a real consensus among the parties, even in less open infrastructure such as the permissioned blockchains.

The employment of the blockchain, allows us to envision new models of governance, where trust to individuals is overcome by consensus from a community. However, it is not yet clear what are the implications in legal terms of this new governance in particular in terms of accountability. More in general, the applicability of the algorithmic governance provided by the blockchain should be better investigated in view of current laws, norms and regulations [9].

Finally, the concept of identity on-chain should be better explored, also in view of the Self-Sovereign Identity [15] concept.

References

- [1] Blockchain Oracles for Hybrid Smart Contracts | Chainlink. <https://chain.link/> [Online; accessed 18. Gen. 2022].
- [2] Tellor. <https://tellor.io/> [Online; accessed 18. Gen. 2022].
- [3] Provable - blockchain oracle service, enabling data-rich smart contracts, 2019. <https://provable.xyz> [Online; accessed 26. Jan. 2022].

- [4] Agency, February 2022. <https://www.law.cornell.edu/wex/agency>, [Online; accessed 28. Apr. 2022].
- [5] Band Protocol - Cross-Chain Data Oracle, 2022. <https://bandprotocol.com/bandchain> [Online; accessed 26. Jan. 2022].
- [6] Hamda Al-Breiki, Muhammad Habib Ur Rehman, Khaled Salah, and Davor Svetinovic. Trustworthy blockchain oracles: review, comparison, and open research challenges. *IEEE Access*, 8:85675–85685, 2020.
- [7] Giulio Caldarelli. Understanding the blockchain oracle problem: A call for action. *Information*, 11(11), 2020.
- [8] Md Sadek Ferdous, Mohammad Javed Morshed Chowdhury, and Mohammad A. Hoque. A survey of consensus algorithms in public blockchain systems for crypto-currencies. *Journal of Network and Computer Applications*, 182:103035, 2021.
- [9] Philipp Hacker, Ioannis Lianos, Georgios Dimitropoulos, and Stefan Eich, editors. *Regulating Blockchain: Techno-Social and Legal Challenges*. Oxford University Press, Oxford, 2019.
- [10] Wulf A Kaal. Blockchain solutions for agency problems in corporate governance. In *Information for Efficient Decision Making: Big Data, Blockchain and Relevance*, pages 313–329. World Scientific, 2021.
- [11] Azienda ospedaliera Bianchi Malacrino di Reggio Calabria. Servizio di manutenzione degli immobili e degli impianti dell'azienda ospedaliera bianchi malacrino di reggio calabria. https://ospedaleri.it/files/old/CSA_manut_%201_%2014_j.pdf [Online; accessed April 2022].
- [12] Diego Pennino, Maurizio Pizzonia, Andrea Vitaletti, and Marco Zecchini. Efficient certification of endpoint control on blockchain. *IEEE Access*, 9:133309–133334, 2021.
- [13] Diego Pennino, Maurizio Pizzonia, Andrea Vitaletti, and Marco Zecchini. Blockchain as iot economy enabler: A review of architectural aspects. *Journal of Sensor and Actuator Networks*, 11(2), 2022.
- [14] Gazzetta Ufficiale. Decreto legislativo 3 aprile 2006, n. 152. "norme in materia ambientale" pubblicato nella gazzetta ufficiale n. 88 del 14 aprile 2006 - supplemento ordinario n. 96. <https://web.camera.it/parlam/leggi/deleghe/06152dl5.htm> [Online; accessed April 2022].
- [15] Fennie Wang and Primavera De Filippi. Self-sovereign identity in a globalized world: Credentials-based identity systems as a driver for economic inclusion. *Frontiers in Blockchain*, 2, 2020.
- [16] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *Ieee Access*, 7:22328–22370, 2019.
- [17] Huanliang Xiong, Muxi Chen, Canghai Wu, Yingding Zhao, and Wenlong Yi. Research on progress of blockchain consensus algorithm: A review on recent progress of blockchain consensus algorithms. *Future Internet*, 14(2):47, 2022.

Blockchain-Based Tracking of the Supply Chain of the Italian Craft Beer Sector^{*}

Lorenzo Ariemma¹, Niccolò De Carlo³, Diego Pennino¹, Maurizio Pizzonia¹,
Andrea Vitaletti² and Marco Zecchini²

¹Università degli Studi Roma Tre, Dipartimento di Ingegneria, Sezione Informatica e Automazione, Via della Vasca Navale 79, 00146 Rome, Italy;

²Sapienza Università di Roma, Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Via Ariosto 25, 00185 Rome, Italy;

³Taggo srl, Viale Giulio Cesare 14, 00192, Rome, Italy;

Abstract

In the Italian craft beer market, many small *breweries* and *pubs* propose a large number of products to *beerlovers*. In this highly fragmented market, it is hard for the beerlovers to assess the quality of a beer and it is hard for breweries and pubs to inform the beerlover of the quality of their offer. Supply chain tracking is an interesting tool to improve transparency of food markets. However, standard tracking approaches are challenging in highly fragmented contexts for several reasons. In this paper, we show how it is possible to realize a blockchain-based supply chain tracking tailored for the highly fragmented sector of the Italian craft beer. We collaborated with one of the players of that market to analyze the specific problems of that context and we report the results of this analysis. We show a design that addresses those problems and might be generalized to support tracking in other highly fragmented sectors. We estimate costs and that turn out to be affordable for that sector.

1. Introduction

In the agriculture and food contexts, the quality of the products is paramount for the final consumer for obvious reasons. However, the consumer has very little means to assess the quality of a product in advance. For this reason auditing and certifications play a crucial role in this context [27]. These approaches rely on the trust that final consumers and market operators pose on a (usually small) number of subjects that perform the auditing. There are two drawbacks to this: the rising of costs, which may put out of market small players, and the undue power given

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

^{*}This research was partially funded by POR FESR LAZIO 2014 – 2020, call for “Gruppi di ricerca 2020”. Det. n. G04052 of Apr. 4th, 2019, under the “LazioChain” project, CUP F85F21001550009 - POR project code A0375E0116. This research was partially funded by Sapienza Ateneo Research grant “La disintermediazione della Pubblica Amministrazione: il ruolo della tecnologia blockchain e le sue implicazioni nei processi e nei ruoli della PA”.

✉ lorenzo.ariemma@uniroma3.it (L. Ariemma); niccolo.decarlo@taggo.io (N. De Carlo); pennino@ing.uniroma3.it (D. Pennino); pizzonia@ing.uniroma3.it (M. Pizzonia); vitaletti@diag.uniroma1.it (A. Vitaletti); zecchini@diag.uniroma1.it (M. Zecchini)

ORCID 0000-0001-8009-4702 (L. Ariemma); 0000-0001-5339-4531 (D. Pennino); 0000-0001-8758-3437 (M. Pizzonia); 0000-0003-1074-5068 (A. Vitaletti); 0000-0002-2280-9543 (M. Zecchini)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

to the auditors, which may improperly change market rules with their decisions and which may be the targets of corruption attempts.

Blockchain technologies provide an opportunity to address this problem differently. These technologies have proven themselves to be valuable tools to address a variety of problems. They are especially useful when a wide number of actors have competing interests but have to cooperate to agree on a common view of some kind of shared data. Historically, the first application of blockchain was cryptocurrency where actors are whoever intend to use the cryptocurrency (e.g., to buy or sell goods) and shared data are the balances of all accounts. Blockchain applications flourished in the last ten years in many contexts [3], like finance, healthcare, logistics, manufacturing, energy, and also agriculture and food. A large part of the efforts in these last two sectors, focused on the adoption of blockchain with the purpose of tracking all the production steps of a supply chain in a decentralized manner [36].

In this paper, we focus on the Italian craft beer sector. A highly fragmented context with a large number of actors and products, and many enthusiast final consumers (*beerlovers*). Since craft producers are by definition small, they can hardly comply with costly certifications and may have difficulties to convey to the final consumer information about the quality of their products. This situation is known to be unfavorable to high quality producers [2] and, hence, detrimental for the whole sector, which ends up with an offer with homogeneously low quality.

The objective of this paper is to report our experience in designing a blockchain-based tracking system for the craft beer supply chain. Since craft beer sector is very fragmented, approaches that can be applied in more consolidated sectors might fail. For example, we cannot ask small breweries an investment to host a node of a dedicated private blockchain. Our analysis was performed with the collaboration of Yhop [39], an Italian startup company that aims to provide tools to improve visibility and product quality awareness for all the actors in the craft beer supply chain, in Italy. The challenges we faced in our work were mostly related to costs. Besides not being able to convince small companies to host nodes, it is equally unaffordable for them to pay the high fees that a common public blockchain may have [31]. Our solution is based on the EOSIO public blockchain. A blockchain that, under certain conditions, can greatly reduce usage costs, and also charge them on a subject that is different from the transaction submitter. Our design is able to leverage EOSIO peculiar features to provide a solution that is affordable for the craft beer sector. We think that the same approach might be adopted in other fragmented sectors, like in the sectors of automotive maintenance, baked food production, or software production.

In Section 2, we present some definitions about the supply chain of craft beer and its problems. In Section 3, we provide the current state of the art involved in our scenario, in particular, in Section 3.1 we show the main approaches for the craft beer market in Italy and in Section 3.2 we provide some information about the adoptions of the blockchain technology for the supply chain. In Section 4, we highlight our objectives to improve the tracking system. In Section 5, we discuss the threat model we face. In Section 6, we present our solution supported by cost analysis. Finally, in Section 7, we provide some conclusions and discuss some future works.

2. The Supply Chain of Craft Beer

In this section, we describe the structure of the supply chain of craft beer with information about involved stakeholders and some of its problems that are relevant for this paper.

Like all products in today's world, beer is driven by new trends. Consumer behavior and pressure, as well as industry consolidation and restructuring, the growth of third-party services providers, and technological developments are also changing the beer supply chain. However, for the sake of simplicity, we can generalize and say that the craft beer supply chain consists of four main stakeholders.

Growers. The chain starts from the growers of raw materials (e.g., hop, malt and yeast). In Italy, it is estimated about 100 hectares [1] cultivated with hops and 2 malt houses [5].

Breweries. The raw materials are then used in the *breweries*. Currently, about 1600 small breweries operate in Italy [28]. They offer services to about 300 of them. Depending on their size, each of them can produce from about 40 to 150 batches each year.

Pubs. All these batches, are then distributed to the dense network of *pubs* throughout the country.

Beerlovers. The last step of the supply chain are represented by the final consumers, i.e., the *beerlovers*.

The craft beer market is fragmented [23] and it cannot be otherwise. In fact, its very definition implies that a multitude of small breweries exists, with scarcely automated production processes. Further, consumers of craft beer are attracted by the variety of offer, which cannot be obtained by big producers with large scale industrial processes.

Any market is more or less affected by an asymmetry of information between the consumer/buyer and the producer/seller. This was the topic of very well-known studies in economics (see for example [2]). Those studies pointed out that when the buyers do not have enough information to perform their choice also the producer is not motivated to propose high quality offers. For big producers, this is mitigated by consumers that can gather and exchange their opinions. This is especially true if products do not change over time and consumers have some way to let know their opinion to other consumers (and are motivated to do that). In the case of highly fragmented markets, as it is the case for the craft beer market, each player is very small and the number of products available to the consumer is very large. Further, products may change frequently due to the enthusiast experimental attitude of each brewer or depending on the availability of ingredients on the market. For this reason, improving the visibility that a consumer may have upfront over the quality of the production is important in the craft beer market.

The lack of accurate information along the chain also spreads from the consumers to the producers. This can cause the so-called *bullwhip effect*. The bullwhip effect is a distribution channel event that refers to shifts in inventory levels due to changes in consumers demand. Demand expectations succumb to supply chain inefficiencies as you go up the supply chain. For example, consumer forecast demand is 25 units, a retail order that includes safety stock becomes 40 units, wholesale order 60 units to gain wholesale purchasing advantages, and manufacturers'

raw material order is 70 units to reduce costs. The bullwhip effect in this scenario creates 45 units more than expected consumer demand. By increasing producer visibility across the entire chain, it can greatly reduce waste, again benefiting the quality of the final product.

Currently, the craft beer production and distribution chain has 3 main limitations.

Inefficiency. Breweries have no tools to manage the production of beer: it can happen that they produce less beer than what is requested or vice versa. In addition, they have no data about the consumption of their products and there are no tools to monitor what beer is preferred by consumers. This may cause the above mentioned bullwhip effect. On the other side, distributors have no tools to manage their inventory; this leads to the risk to lose some barrels due to their expiration.

No guarantees of freshness. Breweries have no possibility to track their barrel. Once they leave the barrels to the distributors, they do not know when and where it is going to be delivered to the pubs/shops. They have no idea how long the beer is going to remain in stock to the distributors and have no guarantee of freshness maintenance during the distribution process.

No product traceability. Currently, there are no systems to track beers' production. Pubs/shops receive the products from the distributors, but they have no information about some important production parameters, such as when it has been produced and how long it stayed to the distributor before its delivery.

By informal discussion with craft beer market operators, supply chain tracking may be perceived as a significant value added.

3. State of the Art

3.1. Tracking in the Craft Beer Sector in Italy

At least part of the breweries feel that the perceived quality of the product may be improved by adopting some supply chain tracking. This is a very common motivation for tracking and especially for tracking by using blockchain (see Section 3.2). In Italy, some experiences are already started from big and small players. Currently, a big Italian player [24] has an experimental project to track the national origin of malt using the Ethereum [42] blockchain. A small brewery [22] is focusing on a centralized tracking of its production batches. Users can access data about the batches on the web by specifying the batch number or by scanning a QR-code on the beer.

For the craft beer sector, Yhop [39] has set up a form of centralized supply chain tracking. It involves breweries and pubs that have signed up with Yhop and allows them to provide information for the beerlovers through the Yhop mobile app. Breweries can provide information about batch productions. Pubs can provide information about what beers they offer and what beers they currently have attached to the taps. Consumers can express their appreciation for the products by highlighting their consumption habits.

Institutions are taking steps to build the Italian brewing chain. The sudden development of this sector made it necessary to fill a regulatory gap in the definition of beer and craft brewery.

With Art. 35 of Law no. 154 of 28 July 2016, a legislative definition of craft beer was provided, thus integrating Law no. 1354 of 16 August 1962, identified as “the product obtained from alcoholic fermentation with *Saccharomyces carlbergensis* or *Saccharomyces Cerevisiae* strains of a must prepared with malt, whether or not roasted, of barley or wheat or their mixtures and water, amaricated with hops or its derivatives or with both”. Art. 35 defines craft beer “beer produced by small independent breweries and not subjected, during the production phase, to pasteurization and microfiltration processes”.

The purpose of the request for Italian raw materials, especially from craft beer producers, is to promote a path that leads to the production of 100% Made in Italy beer, which for the consumer is synonymous with quality, transparency and identity bond with the product. Furthermore, compared to what happens for industrial beer, the consumer of craft products wants to experiment with new aromas, new taste experiences and textures [6]. At the same time, it wishes to rediscover the authenticity of raw materials and establish a link with the territory [37, 4]. Therefore, the area of origin of the product has an identity value and a sense of belonging. This translates into the opening of market niches as many as the preferences expressed by the consumer. Craft beer, with its highly differentiated raw materials, lends itself well to this new trend.

3.2. Blockchain for Supply Chains

Tracking in supply chains is one of the most widespread applications of blockchains. Several systematic literature reviews exist [10, 12, 13, 26, 33]. Further, a number of research works also address the specific problem of food supply chain, see for example [9, 11, 25, 34, 38, 40, 41]. Most of the approaches adopt permissioned blockchains. However, this implies to have

- a number of entities that are willing to provide resources to operate the nodes of the blockchain, and
- a central authority (likely a consortium) that recognizes those subjects as entitled to do that.

This is possible and convenient only in structured ecosystems where some sort of consortium is already present or when the number of involved subjects is small and can coordinate for this purpose. In certain sectors, only a handful of big players can meet the above conditions which may lead to a quite centralized tracking. On the other hand, solutions that rely on public blockchains (like [24]) have to face cost problems. In fact, public blockchain costs may unexpectedly rise due to the adoption of the same blockchain by other applications or communities for reasons that are completely unrelated with the supply chain and are very hard to predict. This is a typical issue when adopting a public blockchain to support a specific business or task. An analysis of these and other problems concerning the interplay of technical and economical aspects can be found in [31].

Further, transaction costs are usually charged on the actor who send the transaction (like in Ethereum [42]). This might be economically inconvenient, since the cost of tracking might be too high, discouraging the supply chain actors to use the tracking service. It might also be unpractical since each actor have to keep an account on the blockchain for the tracking expenses and periodically replenish it.

Some newer unpermissioned blockchains try to lower the fee costs with diverse approaches. Nano [29] is feeless and assumes that nodes operators have other motivation besides revenues for operating the node. However, Nano does not support smart contracts. IOTA [32] asks the users to participate in transaction confirmation. EOSIO [16] adopts an innovative cost model combined with a staking approach, based on paying only for the maximum amount of resources that can be used in a period of time (i.e., memory/RAM, CPU and network), instead of charging for each transaction. It supports smart contracts and adds the possibility of charging to the contract creator the costs of the resources used by contract-related transactions. Other approaches achieve low fees for most transactions (e.g., NEO [30]).

It is worth noting that transforming supply contracts into smart contracts could be a particularly effective remedy to problems such as the above mentioned bullwhip effect. Smart contracts targeted to optimize supply chains are dealt with in [7, 8].

4. Objectives: from Centralized to Blockchain-Based

Yhop [39] operates a centralized platform to connect all stakeholders in the craft beer supply chain, enable them to exchange information efficiently and create an active community around the theme of craft beer. The Yhop platform is the starting point of the solution presented in this paper.

Centralized tracking has some drawbacks. The first and obvious one is that centralized tracking may be considered not secure (see Section 5). However, this may not be the primary concern for the beer sector. Nevertheless, any centralized tracker is responsible for the data that it shows, which may not be desirable for any organization proposing or managing a supply chain tracking system. Currently, Yhop can centrally track craft beer supply chain starting from breweries. A brewery kegs the product and makes it available for distribution or directly to the sales premises to the public. Throughout the supply chain, the information related to the keg up to the consumption of the beer can be recorded. The objective of the project presented in this paper is to use the blockchain to certify this information by cryptographic means and make the actors involved in the process responsible for it. Once the supply chain has been traced in certified form, it will be possible to trace the authenticity of the information presented for each barrel sold at any time and the correctness of the time elapsed between the various stages from production to sale. In addition to giving a guarantee on the authenticity and freshness of the product, this model also guarantees consumers from fraud on the originality of the product itself with respect to counterfeits.

Our first objective is to support blockchain-based tracking in the craft beer supply chain with a reasonable level of security and decentralization. Since the context is made of a large number of small players, our intent is to make tracking adoption as easy and economically convenient as possible. Breweries, beerlovers, and pubs should be able to use the system transparently without the need to explicitly create or manage any blockchain account and without the need to buy cryptocurrency to pay for transactions. At the same time, it should be possible to get enough details to be sure of the integrity of the tracked information, even if this is expected to be asked by a minority of technically-skilled users.

Currently, we intend to focus on the breweries production tracking, but the design should be

compatible with a future involvement of pubs and raw material producers (primarily malt and hop). We also intend to make technology choices that may be used for supporting other services involving tokens targeted to that specific supply chain (e.g., discount coupons or complementary currency).

Our solution should be managed by a subject that decides the architecture, develops the software, deploys it, enrolls users, and supports them so that they can easily perform all operations. We call it the *managing operator*. In our case, the managing operator is Yhop. Currently, the only kind of users involved are breweries, entitled to enter tracking records, and beerlovers, entitled to query tracking records. While enrolment is needed for breweries (and in the future also pubs and raw material producers), beerlovers do not need enrolment, since tracking records are public.

5. Security Goals

We assume the integrity of tracking data to be paramount: the beerlover should be able to get a cryptographic proof that guarantees that data comes directly from the breweries, for the batch that (s)he is interested in. In fact, in this context, data corruption may lead to a serious reputational damage or improper competitive advantage. For this reason, it is important for the data to be cryptographically linked with the brewery so that the brewery is liable for it, both legally and ethically.

The managing operator should not be involved in the security aspects related to data integrity. In this sense, all the users should consider the managing operator as untrusted and the system should allow any user to check the integrity of tracking data at any time. Further, we assume users do not trust any single subject in the supply chain. However, they can trust subjects that are not involved in the supply chain and unlikely to collude with any subject involved in a supply chain, like the nodes of a public blockchain.

Note that, although breweries are legally responsible for stored data, we can not ignore the possibility of a DoS attack. Even non-malicious breweries may inadvertently deplete resources by mistake, for example by re-submitting the same record several times. To prevent this type of attack, we prefer the submission of transactions for the tracking records to be performed by the managing operator (see Section 6), screening our system by misbehaving breweries. Note that, the managing operator still cannot tamper the tracking records, since they are signed by the corresponding brewery, and there is no advantage for the managing operator to stop the submission of tracking records. A possible evolution of the system might include a protocol in which the managing operator commits itself to submit one tracking record at least once, so that a brewery can demonstrate potential malicious behavior by the managing operator. However, this is not dealt with in this paper.

6. A Practical Design

To meet the goals stated in Section 4, we decided to opt for a public blockchain. This allows us to avoid the problem to deploy private nodes. In fact, as described in Section 2, it would be very hard to motivate several small breweries to host nodes. However, as cited in Section 3.2, this

choice has some drawbacks. In particular, the costs of a public blockchain are hard to control and may depend on the load of the system. We opted for the EOSIO blockchain [16] for its peculiar cost and charging model.

6.1. EOSIO Basics

EOSIO (with native token EOS) was started in 2018 by a company named block.one. Its objectives are to provide a platform supporting the development of efficient and cheap-to-execute smart contracts. While it is mostly famous for its very short block time (0.5 seconds), for our application, the most interesting feature is the transaction payment and accounting models. Users put at stake a certain amount of EOS tokens to reserve CPU and NET resources for the execution of transactions, comprising calls to smart contracts. EOSIO automatically *replenish* the available resources favoring accounts showing lower frequency of transactions [21]. The *available resource* (CPU or NET) is the resource still available for consumption supposing no replenish is going to be performed. Persistent memory (called RAM in EOSIO jargon) used by smart contracts have to be bought on a free market but can be sold when not needed. Essentially, resources for EOSIO usage follow the CAPEX model. This means that if smart contract calls are performed at a constant frequency and if smart contracts are designed so that their storage consumption is bounded, the use of EOSIO has no direct operational costs.

Further, in EOSIO, more users (i.e., public keys) can be associated with an account where each user can be assigned specific permissions. Since resources are bound to the account, all users consume resources of that account. The managing operator is also a special user of that account, which can control the available amount of resources for running the smart contract and provision more of them when needed, i.e., in case tracking activity increases.

The permission system of EOSIO works as follows (see Figure 1). We limit this description to what is strictly needed in this paper. Further details can be found in [14]. Each account is associated with many *permissions*. Each permission is associated with many public keys, which in EOSIO jargon are called *authorities*. One smart contract method is called *action* in EOSIO jargon and it is associated with one permission. When an action A is associated to a permission P , only an authority u associated with P can call A . EOSIO natively performs this check by verifying that the transaction containing the call to A is signed by the public key of u . This check is performed before charging available resources. Clearly, a smart contract can perform further checks, but their execution impacts on available resources.

6.2. The Data to Be Tracked

Essentially, our tracking system allows a number of breweries to store in the RAM of a smart contract in the public EOSIO blockchain, a record for each batch of beer they produced. First, we note that we can delete from RAM records for batches that are too old, since there is no point in tracking batches beyond their expiration date. Hence, in a stationary condition, where the number of breweries and their *batch production rate* is constant over time, the amount of storage needed in blockchain is bounded. To further reduce storage, our *tracking records* are the 5-tuple $\langle B, b, h_b, d_e, t_s \rangle$, where B is the identifier of the brewery that has submitted the tracking record, b is the identifier of the batch to track among the batches produced by B , h_b is

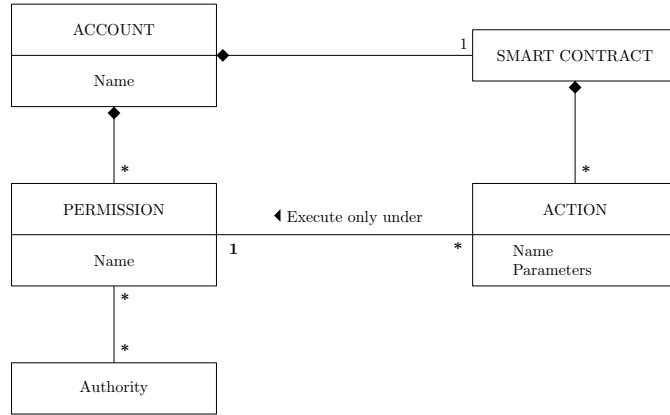


Figure 1: EOSIO Permissions model.

a cryptographic hash of the record describing the batch, d_e is the expiration date of the batch, t_s is the timestamp of the submission of this tracking record. As breweries identifiers, we can choose its public key or we can assign smaller integer numbers to them. Each batch is described by a *batch description*, which is a json-represented record containing all data specified by the brewery. Yhop already collects batch description and stores them in its systems. The hash h_b should be a hash of the batch description. Unfortunately, a json representation is not unique. For this reason, a canonicalization scheme, like the one described in [35], should be adopted before applying the cryptographic hash function.

6.3. The Architecture

The architecture is shown in Figures 2 and 3. Since the only subject that have to pay for resource consumption is the managing operator, breweries have not specific accounts. They are just represented by their public keys that are associated with the *tracking* permission. The *active* permission is the default EOSIO permission which is dedicated to the managing operator.

The smart contract has an action *add*, which can be called by breweries to add tracking records. This call could be in principle be performed by clients used by breweries by directly submitting the corresponding transaction to the blockchain nodes. However, even non malicious users may inadvertently deplete resources by mistake, for example, by repeatedly clicking on the submit button and hence re-submitting the same record multiple times. To filter out these kinds of mistakes, we prefer to make the managing operator servers to actually perform the submission of the transaction. We call this service *tracking support*, also denoted S in the following. Note that, since the transaction is signed by a brewery, the tracking support service cannot modify it. In principle, the managing operator can block the submission, but this is not in its interest unless it is a repeated submission. The tracking support service also performs further checks to be sure that submitted tracking records are consistent with the corresponding batch description, in particular regarding expiration date and hash.

The smart contract stores tracking records in a so-called *kv-table* [17], whose size is accounted as occupied RAM. In EOSIO, a kv-table is made of rows indexed by a key. A kv-table belongs to

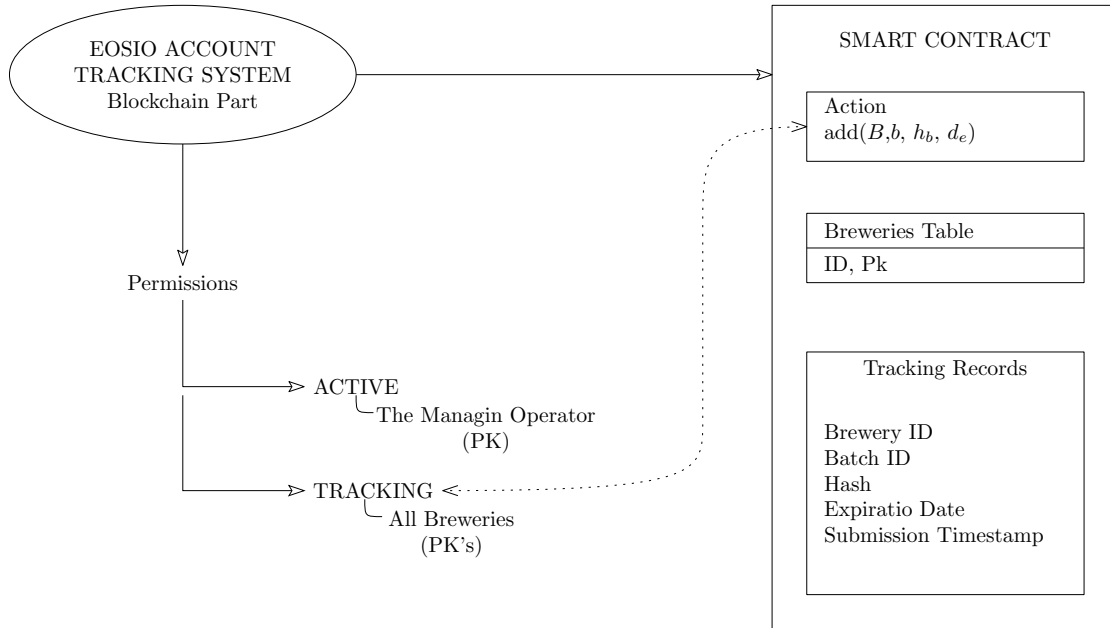


Figure 2: Architecture: on-chain part.

a smart contract and can be queried by an off-chain client using standard node REST API.

6.4. Lifecycle of a Tracking Record Submission

The steps to submit a tracking record for a batch into our blockchain-based system are the following. Please refer to Figure 3.

- 1 The brewery owner (whose brewery has ID B) has a batch description for a (new) batch (with ID b).
- 2 The Brewery Client C represents the batch description in a canonized json and send it to off-chain system for storage.
- 3 C obtains a cryptographic hash h_b from the canonized json.
- 4 C creates the transaction tx for the call of the *add action*.
- 5 C signs the transaction tx with the secret key sk_B of the brewery and send it to the Tracking Support Service S .
- 6 S stores signed tx .
- 7 S checks that parameters in tx are coherent with the corresponding batch description already stored in the off-chain systems and that this request was not already processing (case of repeated unintended submissions).

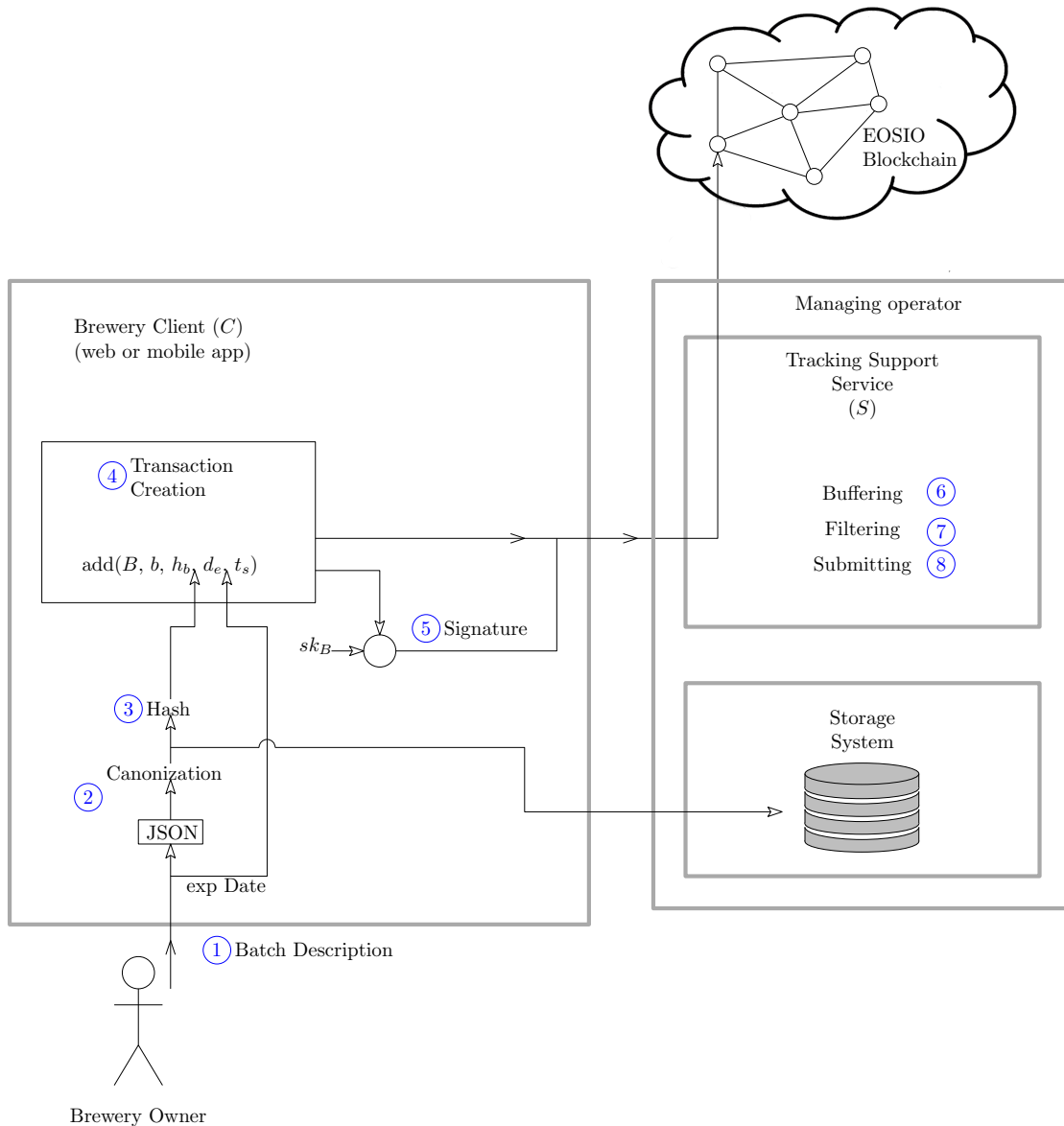


Figure 3: Architecture: off-chain part.

- 8 S submits tx to any public EOSIO nodes.
- 9 S checks tx for confirmation.
- 10 S provides the user with feedback on the result of the add action. This is made by any asynchronous communication mean (e.g., by email).

6.5. Cost Estimation

We performed some preliminary experiments to estimate the resources needed to run the system and to devise a methodology to make this estimation when the final version of smart contract will be ready for production.

We created a very simple smart contract that have a multi-index kv-table, and an action *insert* to insert a preliminary form of tracking records in it. For this experiment, our tracking record has the following structure $\langle id_1, id_2, h \rangle$, where id_1 is an integer, id_2 is a 7 character string, and h is a 32 character string. In our experiment, each insertion takes 156 bytes of RAM and $300\mu s$ of CPU to execute the insert action.

Following the number given in Section 2, we suppose to have 300 breweries managed by Yhop, each inserting at most 150 tracking records per year, which gives a maximum frequency of 45,000 insertions of tracking records per year. Since the expiration time of a batch of beer is 1 year, and we do not need to track expired batches, 45,000 is also a good upper bound of the maximum number of tracking records that our smart contract will need to store to support tracking for the current breweries coverage of Yhop.

We estimate storage costs as follows. The total storage consumption (RAM) is of about 7020Kbytes. At the time of writing the RAM costs about 0.029 EOS per Kbyte [19], for a total cost of 203 EOS, currently corresponding to about 400 euro. Note that these are CAPEX.

NET and CPU resources are managed in a similar way in EOSIO. In the following, we focus on the CPU cost (NET cost turns out to be negligible). As we describe in the following, the CAPEX model is viable form a theoretic point of view but very expensive in practice and for the CPU resource we need to choose an OPEX approach supported by the EOS *powerup* tool [15].

EOSIO keeps an amount of *available CPU* for each account that is regularly replenished. The description of the replenishing algorithm given in the EOSIO documentation [21] is not very clear. However, we checked it with the code [18] and we observed that it takes 24 hours (called *window*) to fully replenish the available CPU (and NET) resources. Following what we stated above, we assume 45,000 tracking records per year and 200 working days per year. Further, assuming tracking transactions to be homogeneously distributed, we obtain $45,000/200 = 225$ transactions per day. Assuming tracking activity to be uniformly distributed within 10 hours of work for each day, we obtain a transaction every about 2.6 minutes, on average and a rest period of 14 hours. The transaction frequency during working hours is very high, so we cannot expect any notable replenishment there, and the rest hours are not enough for completely replenishing the CPU (or NET) resource. Further, the replenish rate is not dependent on the staked amount of resources. Hence, as far as we can tell, there is no way to take advantage of the EOSIO staking model to support our transaction frequency with a single account. To work around this oddity, we might tweak the architecture shown in Section 6, so that transactions are originated from three distinct accounts, which are equally configured regarding permissions and authorities. We might use each of them to send transactions for more than 12 hours (e.g., 13 hours) in a round robin fashion. In this way, one account can completely rest for more than 24 hours, to be completely replenished, while the other two work. This makes the CAPEX approach theoretically viable also for CPU and NET resources. In practice, each of the three accounts should stake CPU time to execute 225 transactions, corresponding totally to about 200ms of CPU currently costing about 80,000 euros. Even if these are capital expenses, this is

clearly a very high cost.

Recently EOSIO has introduced a way to rent unused staked resources called *powerup* model [15]. From the point of view of the managing operator, the powerup model is essentially a way to buy resources under the traditional OPEX model, which is currently very cheap (0.0002 EOS for each millisecond of CPU [20]). In this model, 225 transactions per day can be executed by spending about 0.10 EUR (at current exchange rates), for a cost of less than 40 euro per year. In conclusion, for the CPU resource the OPEX model using powerup is definitely more advisable. The same approach can be adopted for the NET resource, with even lower costs.

7. Conclusions and Future Work

In this paper, we show how it is possible to design a tracking system based on the EOSIO blockchain that is affordable for the fragmented sector of the Italian craft beer.

To assess the truthfulness and integrity of the tracking information, and to give responsibility to the actors that created the data, the blockchain is used to store the hashes of the records that describe production batches. The batch descriptions are stored in a centralized way on the servers of the managing operator (i.e., Yhop in our use case). We give a cost estimation of our solution and we show that, at current prices, mixing CAPEX and OPEX cost models is the most convenient approach.

As a future work, we plan to realize the system and to perform a proof-of-concept in collaboration with Yhop and with more than twenty breweries that already have declared their availability to participate. We plan to extend the functionalities of the system to include pubs and growers. We also intend to study how to use blockchain in the same context to support discount coupons or a complementary currency.

8. Acknowledgements

We would like to thank you Lorenzo Cannella for working on an early version of this project.

References

- [1] Agronotizie. Luppolo made in italy un panorama produttivo vivace ma frammentato. [online]. <https://agronotizie.imagelinenetwork.com/agricoltura-economia-politica/2020/08/31/luppolo-made-in-italy-un-panorama-produttivo-vivace-ma-frammentato/67793>, last viewed April 2022.
- [2] George A. Akerlof. The Market for “Lemons”: Quality Uncertainty and the Market Mechanism. In Peter Diamond and Michael Rothschild, editors, *Uncertainty in Economics*, pages 235–251. Academic Press, January 1978.
- [3] Jameela Al-Jaroodi and Nader Mohamed. Blockchain in industries: A survey. *IEEE Access*, 7:36500–36515, 2019.
- [4] Tiziana Amoriello, Katya Carbone, Alessandro Monteleone, Mauro Pagano, and Serena Tarangiol. Criticità e opportunità per lo sviluppo sostenibile della filiera brassicola. [online]. https://www.academia.edu/30444023/CRITICIT%C3%80_E_OPPORTUNIT%C3%80_

- PER_LO_SVILUPPO_SOSTENIBILE_DELLA_FILIERA_BRASSICOLA, last viewed May 2022.
- [5] AssoBirra. Annualreport 2020 june 2021. [online]. https://www.assobirra.it/wp-content/uploads/2021/06/AssoBirra_AnnualReport_2020_giugno2021_DEF.pdf, last viewed April 2022.
 - [6] Bram Berkhout, Lianne Bertling, Yannick Bleeker, Walter de Wit, Geerten Kruis, Robin Stokkel, and Ri-janne Theuws. The Contribution made by Beer to the European Economy | FullReport. [online]. <https://brewersofeurope.org/uploads/mycms-files/documents/archives/publications/2013/FullReport20140123.pdf>, last viewed May 2022.
 - [7] Paolo Bottoni, Claudio Di Ciccio, Remo Pareschi, Nicola Gessa, and Gilda Massa. Variants in managing supply chains on distributed ledgers. *arXiv preprint arXiv:2205.06766*, 2022.
 - [8] Paolo Bottoni, Nicola Gessa, Gilda Massa, Remo Pareschi, Hesham Selim, and Enrico Arcuri. Intelligent smart contracts for innovative supply chain management. *Frontiers in Blockchain*, 3:52, 2020.
 - [9] Miguel Pincheira Caro, Muhammad Salek Ali, Massimo Vecchio, and Raffaele Giaffreda. Blockchain-based traceability in Agri-Food supply chain management: A practical implementation. In *2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*, pages 1–4. IEEE, 2018.
 - [10] Shuchih E. Chang and Yichian Chen. When blockchain meets supply chain: A systematic literature review on current development and potential applications. *IEEE Access*, 8:62478–62494, 2020.
 - [11] Jiang Duan, Chen Zhang, Yu Gong, Steve Brown, and Zhi Li. A content-analysis based literature review in blockchain adoption within food supply chain. *International journal of environmental research and public health*, 17(5):1784, 2020.
 - [12] Davor Dujak and Domagoj Sajter. Blockchain applications in supply chain. In *SMART supply network*, pages 21–46. Springer, 2019.
 - [13] Pankaj Dutta, Tsan-Ming Choi, Surabhi Somani, and Richa Butala. Blockchain technology in supply chain operations: Applications, challenges and research opportunities. *Transportation research part e: Logistics and transportation review*, 142:102067, 2020.
 - [14] EOSIO. Accounts And Permissions | EOSIO Developer Docs. https://developers.eos.io/welcome/latest/protocol-guides/accounts_and_permissions.
 - [15] EOSIO. The EOS PowerUp Model Explained – EOSIO. <https://eos.io/news/eos-powerup-model-explained>.
 - [16] EOSIO. Eosio Documentation. [online]. <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>, last viewed April 2022.
 - [17] EOSIO. How-To Use Key-Value Table | EOSIO Developer Docs. https://developers.eos.io/manuals/eosio.cdt/latest/how-to-guides/key-value-api/kv_table/how-to-use-kv-table.
 - [18] EOSIO. Nodeos code, master branch at 30 april 2022. [online]. https://github.com/EOSIO/eos/blob/11d35f0f934402321853119d36caeb7022813743/libraries/chain/include/eosio/chain/resource_limits_private.hpp#L110, https://github.com/EOSIO/eos/blob/11d35f0f934402321853119d36caeb7022813743/libraries/chain/resource_limits.cpp#L166, https://github.com/EOSIO/eos/blob/11d35f0f934402321853119d36caeb7022813743/libraries/chain/resource_limits.cpp#L185,

- <https://github.com/EOSIO/eos/blob/11d35f0f934402321853119d36caeb7022813743/libraries/chain/include/eosio/chain/config.hpp#L51> last viewed 30 April 2022.
- [19] EOSIO. Official EOS Explorer and Wallet of EOS Authority | EOS Authority. <https://eosauthority.com/>.
- [20] EOSIO. PowerUp Calculator | EOS Authority. <https://eosauthority.com/power/calculator?network=eos>.
- [21] EOSIO. Staking on EOSIO-based blockchains | EOSIO Developer Docs. <https://developers.eos.io/manuals/eosio.contracts/latest/key-concepts/stake/#system-resources-replenish-algorithm>.
- [22] Giornale della Birra. BeerLife: grazie alla blockchain, tutta la vita della birra che si sta gustando a portata di click! - Giornale della Birra. <https://www.giornaledellabirra.it/senza-categoria/beerlife-grazie-alla-blockchain-tutta-la-vita-della-birra-che-si-sta-gustando-a-portata-di-click/>.
- [23] Indeed. What Is a Fragmented Market? | Indeed.com. <https://www.indeed.com/career-advice/career-development/fragmented-market>.
- [24] Ledger Insights. Birra Peroni uses EY OpsChain, NFTs for beer traceability - Ledger Insights - enterprise blockchain. <https://www.ledgerinsights.com/birra-peroni-uses-ey-opschain-nfts-for-beer-traceability/>.
- [25] Andreas Kamilaris, Agusti Fonts, and Francesc X. Prenafeta-Boldo. The rise of blockchain technology in agriculture and food supply chains. *Trends in Food Science & Technology*, 91:640–652, 2019.
- [26] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. Digital supply chain transformation toward blockchain integration. In *proceedings of the 50th Hawaii international conference on system sciences*, 2017.
- [27] Konstantinos V. Kotsanopoulos and Ioannis S. Arvanitoyannis. The role of auditing, food safety, and food quality standards in the food industry: A review. *Comprehensive Reviews in Food Science and Food Safety*, 16(5):760–775, 2017.
- [28] Microbirrifici. map of active producers. [online]. https://www.microbirrifici.org/beer_italy_maps.aspx, last viewed April 2022.
- [29] Nano. Eco-friendly and feeless digital currency. [online]. <https://nano.org/>, last viewed January 2022.
- [30] Neo. Whitepaper. [online]. <https://docs.neo.org/v2/docs/en-us/basic/whitepaper.html>, last viewed January 2022.
- [31] Diego Pennino, Maurizio Pizzonia, Andrea Vitaletti, and Marco Zecchini. Blockchain as IoT Economy Enabler: A Review of Architectural Aspects. *Journal of Sensor and Actuator Networks*, 11(2):20, 2022.
- [32] Serguei Popov. The tangle. *White paper*, 1(3), 2018.
- [33] Maciel M. Queiroz, Renato Telles, and Silvia H. Bonilla. Blockchain and supply chain management integration: a systematic review of the literature. *Supply Chain Management: An International Journal*, 2019.
- [34] Michael Rogerson and Glenn C. Parry. Blockchain: case studies in food supply chain visibility. *Supply Chain Management: An International Journal*, 2020.
- [35] Anders Rundgren, Bret Jordan, and Samuel Erdtman. JSON Canonicalization Scheme (JCS). Request for Comments RFC 8785, Internet Engineering Task Force, June 2020.
- [36] Samant Saurabh and Kushankur Dey. Blockchain technology adoption, architecture, and

- sustainable agri-food supply chains. *Journal of Cleaner Production*, 284:124731, 2021.
- [37] Steven M. Schnell and Joseph F. Reese. Microbreweries as tools of local identity. *Journal of cultural geography*, 21(1):45–69, 2003.
- [38] Affaf Shahid, Ahmad Almogren, Nadeem Javaid, Fahad Ahmad Al-Zahrani, Mansour Zuair, and Masoom Alam. Blockchain-based agri-food supply chain: A complete solution. *IEEE Access*, 8:69230–69243, 2020.
- [39] Taggo srl. Yhop. [online]. <https://www.yhop.beer/>, last viewed April 2022.
- [40] Bowen Tan, Jiaqi Yan, Si Chen, and Xingchen Liu. The impact of blockchain on food supply chain: The case of walmart. In *International Conference on Smart Blockchain*, pages 167–177. Springer, 2018.
- [41] Feng Tian. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In *2016 13th international conference on service systems and service management (ICSSSM)*, pages 1–6. IEEE, 2016.
- [42] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.