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

Stefano De Angelis[1,*], Gilberto Zanfino[1], Leonardo Aniello[1], Federico Lombardi[1,2] and Vladimiro Sassone[1]

[1]University of Southampton, University Rd, Southampton SO171BJ, UK
[2]Conio 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)

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 $N \geq 3f \quad 1$ 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.

(I_2) 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*.

(I_3) 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.

(I_4) 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.

(I_5) 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.

(I_6) 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.

(I_7) 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.

(I_8) 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.

(I_9) 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 Hyperldger Fabric are not affected by this issue since they do not use gas.

(I_{10}) 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.

(I_{11}) 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.

(I_{12}) 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

	PoW	*CPoS*	*PPoS*	*PBFT*	*PoA*
persistency	eventual	eventual	yes	yes	eventual
termination	yes	yes	eventual	$n \geq 2f \quad 1$	eventual
validator fairness	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 *persistency* 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. *Persistency*: 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 persistency is violated after a certain threshold (i.e. confirmation time), it will never be satisfied again. Persistency 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 persistency.
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 persistency, because of the possibility of having forks. However, such forks will eventually be resolved, according to the specifics of the platform. For this reason, persistency must be classified as *eventual*. Conversely, the PPoS protocol does not allow forks, and blocks are instantly finalised, prioritising persistency 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

	Bitcoin	*Ethereum 2.0*	*Algorand*	*Hyperledger*	*Ethereum-private*
confidentiality	no	no	co-chains	channels	private txs
integrity	majority of hash power	majority of stake	majority of stake	up to $3f$ 1	eventual
availability	yes	yes	yes	up to $2f$ 1	eventual
accountability	partial	partial	partial	yes	yes
authorisation	no	no	no	yes	yes
client fairness	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. Mythril. https://github.com/ConsenSys/mythril.

[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.