Axis: A Reliable Blockchain Technology for Real-World Decentralized Applications

Luiz Carvalho

luizcarvalhohen@gmail.com

Abstract. In this whitepaper, we propose "Axis" as an underdevelopment blockchain project that will provide a secure, metamorphic, and reliable blockchain environment to address real-world challenges in real-world applications, and it is equipped with its own particular programming language, which is called "Axis Lang".
Axis is a truly reliable, secure, and efficient method to implement industrial, public ledgers that contribute fork-protection, fairness, and liveness in the blockchain. Axis also follows a carbon-free mining protocol that makes it a perfect choice for next-generation decentralized applications. Finally, our evaluation results prove the usefulness and performance of Axis for large-scale decentralized applications in different domains.

1 Introduction

Blockchain is a distributed ledger technology that designates exchanges of value between individuals securely, permanently, and in a simply provable manner [12]. It is also the underlying technology for cryptocurrencies such as Bitcoin and Ethereum. Even though initially used for financial transactions, applications of blockchain extend beyond finance and can affect a wide variety of industries such as 5G and beyond networks [16]. For example, smart contracts can enable applications to communicate with Things in the IoT [9], in a way similar to how hardware drivers allow applications to cooperate with devices. Moreover, smart contracts can provide high security for the 5G networks involved in decentralized ledgers.

The programmability of the Ethereum platform is predicated on its ability to build and perform smart contracts [12]. The term "smart contract" was introduced by Nick Szabo in 1996 [24], when he described it as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises". Smart contracts are agreements between transacting parties written using computer code and programmed to self-execute when specific conditions are met. These can be integrated into a blockchain platform like Ethereum to implement the verification and integrity required for such an automated system to work.

However, creating trustworthy and secure smart contracts can be remarkably complicated due to the complex semantics of the underlying domain-specific

languages and their testability. There have been high-profile incidents suggesting that specific blockchain smart contracts could accommodate various code-security vulnerabilities, which can potentially lead to financial harm [18]. This is especially challenging given the notion that smart contracts are supposed to be "immutable". In other words, once a contract code is deployed, it cannot be changed anymore, which makes patching identified vulnerabilities impossible.

This proposal introduces **"Axis"** as a next-generation distributed ledger that provides a safe, fast, and inexpensive foundation for a broad spectrum of real-world application domains such as software engineering, telecommunication, energy supply firms, health-care, blockchain startups, technology developers, financial institutions, national governments, and academic communities. In this regard, numerous innovative aspects of the Axis blockchain, such as extensibility, interoperability, security, and flexibility, will identify our blockchain foundation as a concrete resolution that brings significant benefits and innovation to the decentralization ecosystem and its community. Axis also promises a transparent, tamper-proof, and reliable system that can facilitate new business solutions in the marketplace, particularly in combination with its unique LLVM-based compiler called

2 Background

Blockchain is a growing list of linked records [8], named blocks connected and secured using encryption algorithms [20]. The key to this list's effectiveness is the links created from one block to the next, thus making it difficult to change any block after adding it to the list. Hence, we get the name "blockchain" as it is virtually a chain of blocks of data.

This list represents a protected online registry for stating some agreed-on and conducted transactions among different entities or organizations. The recorded transactions are usually generated due to specific activities such as financial, business, industrial, or system activities. The blocks that store the transactions are generally timestamped, encrypted, and replicated on multiple sites and cannot be altered. A group of people or organizations on a network can use blockchain to record some transactions (activities) among them.

The group members can generally review the previously recorded transactions; however, none of them can modify or remove any of the previously registered transactions. This makes blockchain maintain an immutable history of the activities of the group's members. This history is shared among all or selected group members. This offers high levels of traceability of any and all recorded transactions and transparency that allows everyone involved to view these transactions. Yet, it also provides assurances that these records (or blocks) cannot be altered by anyone in the group who created it or anyone else, for that matter. The group agrees upon the logical links created between the transactions; yet, they are irreversible, thus making it impossible to change. One of the main features that were introduced with blockchain is the enabling of two or more entities to securely record an agreement of specific actions over a public network such as

the Internet without including a third party like an authorization entity or government office. The involved entities may or may not know each other, and they do not even have to trust each other. Yet, they can still make the agreement, document it, and have that transaction record appended to the chain. Hence, the record of the agreement after it is appended to the chain cannot be altered, canceled, or denied by any of the entities involved in the agreement.

2.1 Mining

A process called "mining" [26] is used to guarantee the validity and consistency of the conducted agreements appended to the chain. This critical feature was not available before introducing the blockchain. Therefore, blockchain is the main enabler of the Internet of Transactions [3], [4], which is needed to support many industrial applications. Blockchain is an evolving technology initially invented in the Bitcoin arena; however, it is applicable in many industrial applications, some of which we studied and discussed in our white paper.

One of the strong believers of blockchain and its viability in the general business and industrial domain is IBM, as they are investing heavily in the field and working on various enhancements and applications [6]. Before we start to discuss the internals of Axis Blockchain, we will discuss the fundamental features and functions of blockchain data structures that are the central features of industrial applications.

2.2 Blockchain Categories

Blockchain technologies can be roughly classified into three classes (Figure 1) as follows:

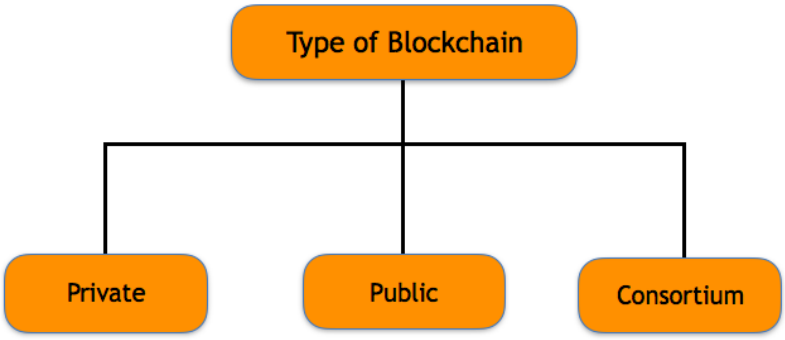

Fig. 1: Different types of blockchains

1. **Public blockchain:** Everyone can check the transaction, verify it, and participate in reaching consensus. Bitcoin and Ethereum are both Public Blockchain. Figure 1 represent SAT Solvers [22] for Bitcoin mining.

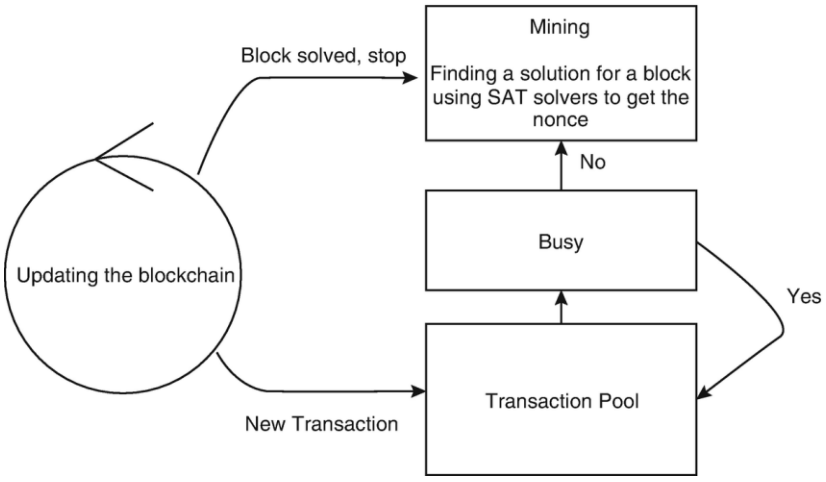

Fig. 2: SAT solver is an algorithm for establishing satisfiability

2. **Consortium blockchains:** In this case, the node with authority can be chosen in advance, usually has partnerships similar business to business, the data in blockchain can be open or private, can be seen as Partly Decentralized. Figure 3 represent a solution to stock exchange based on consortium blockchain [28]. Note that Hyperledger [5] is a successful example of consortium blockchains.

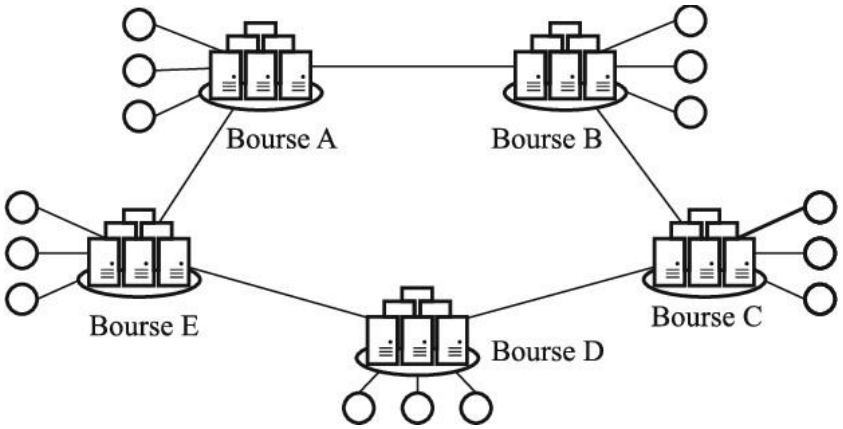

Fig. 3: A solution for stock exchange applications based on consortium blockchains.

3. **Private blockchain:** In private blockchain networks, nodes are restricted to not participate in this blockchain. There is also strict authority management on data access. Even though public blockchains are more well-known in public because of their convenience, sometimes we may need remote control such as consortium blockchains or private blockchain, depending on what service we offer or its community.

   Figure 4 represent how private blockchains can be used finance section.

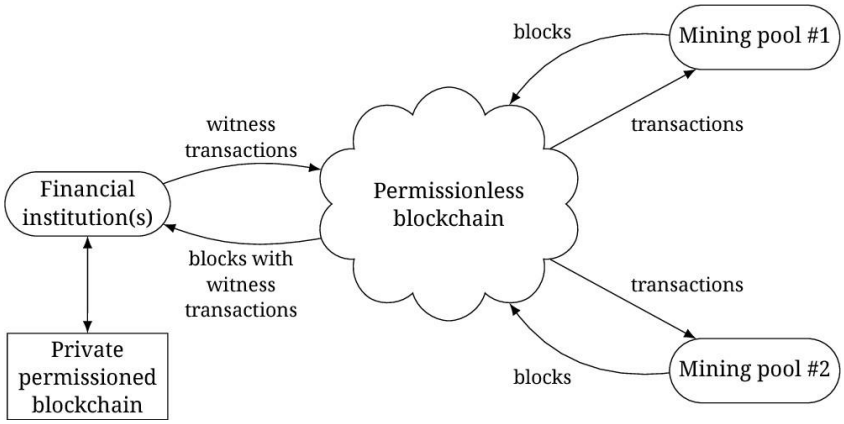

Fig. 4: Example of a private blockchain in the finance section.

2.3 Smart Contracts

Smart contracts are only controlled by code that can handle transactions fully autonomously. Moreover, smart contract code is executed when a user submits a transaction along with a smart contract as the recipient. Users add payload data in transactions, which in turn is provided as input to the subject smart contract. More specifically, a contract is established as a collection of functions, which users can invoke. A contract can also trigger the execution of another contract through *CALL* instruction. This critical instruction transfers a message similar to Remote Procedure Call (RPC) in other programming paradigms [6].

In order to execute a smart contract, a sender has to send a transaction to the subject contract and pay a charge, which is called "GAS" (it will be acquired from the contract's computational cost.). The contracts themselves can also call other contracts present on the Ethereum blockchain [18]. Note that every contract is tied to an account and the contract code can be triggered by calls or transactions received from other contracts. However, accounts cannot launch new transactions on their own, which means they can only respond to other transactions they receive. Since smart contracts are generally designed to manipulate and hold funds designated in Ether, they are considered to be highly attractive targets for cybercriminals [19].

2.4 Smart Contract Vulnerabilities

There are multiple well-known security issues reported in the smart contract ecosystem that all have been comprehensively described in various references such as [4, 18]. However, we briefly introduce some of the most prevalent vulnerability classes that we frequently mention throughout this paper. For the sake of saving space, we present a two-letter acronym for each vulnerability.

Integer Overflow (IO) & Underflow (IU). Integer overflow (and underflow) is a common error in numerous programming languages but in the context of Ethereum it can have serious outcomes. In Solidity "Integer" data types have no built-in security against integer overflow (IOF) and underflow (UOF) attacks [10, 25]. For example, if a loop counter were to overflow, generating an infinite loop, the funds of a contract would become fully frozen. Thus, attackers can exploit this bug by increasing the number of iterations of a loop, for example, by introducing new users to a vulnerable contract [25].

Re-Entrancy (RE). This is a well-known attack that has taken Ethereum security communities by storm, particularly after the notorious DAO hack [23]. This vulnerability will be exploited when a contract attempts to send Ether before having updated its internal state. If the target address is a different contract, the contract code will be executed and can invoke the function to ask Ether again and again, which results in generating funds.

Unhandled Exceptions (UE). Some low-level operations in Solidity (e.g. *send*), which is used to transfer Ether, do not throw an exception on failure, instead they report the status by returning a Boolean. If this returns value were

to be unchecked, a contract would continue its execution even if the payment failed, which could lead to inconsistencies [15].

Transaction Order Dependency (TOD). In Ethereum, different transactions are carried in a single block, which means that the state of a contract can be updated many times in the same block. If the order of two transactions calling the same contract changes the final outcome, adversaries can exploit this property. For example, in the case of a smart contract that expects members to submit the resolution to a puzzle in exchange for a bonus, an adversary member could decrease the amount of the bonus when the transaction is submitted.

Locked Ether (LE). Ethereum smart contracts can also have a function labelled as payable that allows the contract to receive Ether and to increase its balance. The contract can also have a function which sends Ether. For example, a contract might have a payable function called *deposit*, which receives Ether, and a function called *withdraw*, which sends Ether. However, there are several reasons why the withdraw function may become unable to send funds any longer. One reason could be that the contract may depend on another contract which has been destructed using the *SELFDESTRUCT* instruction of the EVM — i.e. its code has been removed and its funds transferred. It is also possible that the withdraw function requires an external contract to send Ether. However, if the dependence contract has already been destructed, the *withdraw* function will not be able to actually send the Ether anymore and lock the funds of the contract. This case occurred in the Parity Wallet bug in November 2017, which resulted in a loss of millions of USD worth of Ether [17].

3 Axis Preliminary Implementation

Axis is still under the development, Listing 3 represents a part of our stack machine. Furthermore, we have built a security model for the automatic security analysis in our blockchain virtual machine.

```
//! Axis VM stack



#[cfg(not(feature = "std"))]
use alloc::vec::Vec;

use bigint::M256;
use super::errors::OnChainError;

/// Represents an AxisVM stack.
#[derive(Debug)]
pub struct Stack {
    stack: Vec<M256>,
}
```

```
impl Default for Stack {
    fn default() -> Stack {
        Stack {
            stack: Vec::new(),
        }
    }
}

impl Stack {
    /// Check a pop-push cycle. If the check succeeded, `push`, `pop`,
    /// `set`, `peek` within the limit should not fail.
    pub fn check_pop_push(&self, pop: usize, push: usize) -> Result<(),
        OnChainError> {
        if self.len() < pop {
            return Err(OnChainError::StackUnderflow);
        }
        if self.len() - pop + push > 1024 {
            return Err(OnChainError::StackOverflow);
        }
        Ok(())
    }


        /// Pop a value from the stack.
        pub fn pop(&mut self) -> Result<M256, OnChainError> {
            match self.stack.pop() {
                Some(x) => Ok(x),
                None => Err(OnChainError::StackUnderflow),
            }
        }

    /// Push a new value to the stack.
    pub fn push(&mut self, elem: M256) -> Result<(), OnChainError> {
        self.stack.push(elem);
        if self.len() > 1024 {
            self.stack.pop();
            Err(OnChainError::StackOverflow)
        } else {
            Ok(())
        }
    }



    /// Peek a value at given index for the stack, where the top of
    /// the stack is at index `0`. If the index is too large,
    /// `StackError::Underflow` is returned.
    pub fn peek(&self, no_from_top: usize) -> Result<M256, OnChainError>
        {
        if self.stack.len() > no_from_top {
```

```
            Ok(self.stack[self.stack.len() - no_from_top - 1])
        } else {
            Err(OnChainError::StackUnderflow)
        }
    }


    /// Set a value at given index for the stack, where the top of the
    /// stack is at index `0`. If the index is too large,
    /// `StackError::Underflow` is returned.
    pub fn set(&mut self, no_from_top: usize, val: M256) -> Result<(),
        OnChainError> {
        if self.stack.len() > no_from_top {
            let len = self.stack.len();
            self.stack[len - no_from_top - 1] = val;
            Ok(())
        } else {
            Err(OnChainError::StackUnderflow)
        }
    }



    /// Get the current stack length.
    #[inline]
    pub fn len(&self) -> usize {
        self.stack.len()
    }

    /// Returns true if stack is empty
    #[inline]
    pub fn is_empty(&self) -> bool { self.len() == 0 }
}
```

3.1 Optimized & Secure Development with Axlang

In Axis's design, we are working on a specific compiler that will enable our developers to write, debug, test, and deploy their secure smart contracts with high performance. We call our language **Axlang**. This compiler will be a next-generation full-stack programming language for secure blockchain developments [14] with solid security and verification choices. Thanks to our LLVM [11] native-code compiler, Axlang will serve from the blockchain low-level machine instructions to high-level scripting and even provides user-interface development for decentralized application designs. Hence, a Axlang smart contract [29] will not need to incorporate with any third-party technologies such as web3.js [13] or truffle to be disposable in public.

In other words, the implementation opportunities in Axlang intend to create a full-stack programming language so that developers no longer need to struggle

with various sophisticated technologies and deal with potential inconsistencies. As a result, Axlang accelerates DApp development drastically by aggregating all required stack layers into one unified interface. Furthermore, Axlang generates secure and reliable low-level instructions by incorporating static type checking, static code analysis, formal verification, and runtime taint tracking peculiarities. As we represented the core structure of Axlang in Figure 5, our compiler works based on the following steps:

1. **User Code**
2. **Language Libraries**
3. **Formal Verification**
4. **Memory Manager**
5. **Lexical Parser**
6. **Code Analyzer (sub-component of Lexical Parser**
7. **Dynamic Control & Instrumentation Agent**

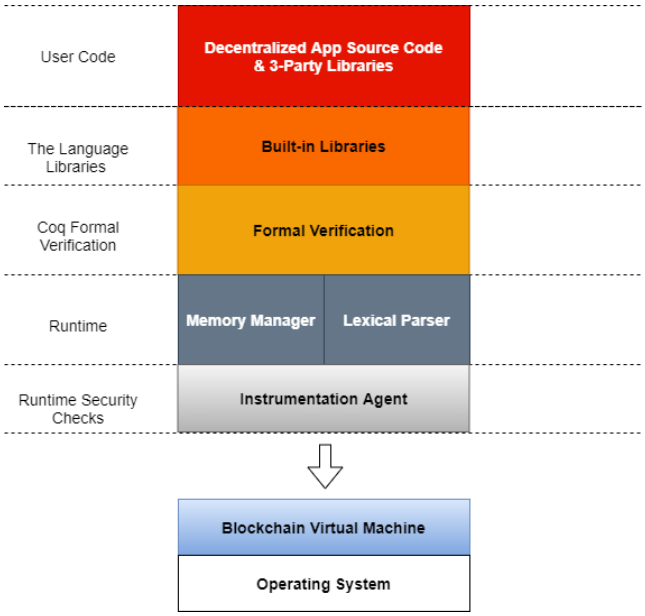

Fig. 5: The abstract structure of Axlang

3.2 Automated Security Audit in Axis Smart Contracts

Critical Operation. To have a better understanding of the security exploitation in the smart contract ecosystem, we studied all reports available on the National

Vulnerability Database (NVD) in order to extract and specify the most critical low-level instructions in the current blockchain work that are commonly involved in cyber attacks. As a result, we concluded that there are a number of low-level instructions (in the EVM) involved in most of the exploits that are essentially linked to value transformation operations. For example, creating transactions (*CALL*), transaction termination (*SELF-DESTRUCT*), code injections (*CALL CODE*), and (*DELEGATECALL*) are some of the most repeated instructions that the public exploits databases. Listing 1.1 represents some of the critical instructions in the abstract level in a smart contract ecosystem, and Table 1 shows the details of some of the most reported sensitive instructions.

Listing 1.1: A sample generated exploit (for easier reading the sample contract is shown at the source level)

```
<address>.call(bytes memory) returns (bool, bytes memory)
<address>.delegatecall(bytes memory) returns (bool, bytes memory)
<address>.staticcall(bytes memory) returns (bool, bytes memory)
```

Accordingly, we found that smart contract attackers often steal tokens by exploiting these critical instructions or in some cases, they attempt to interrupt target contracts by triggering errors in the code logic. Consequently, to implement our security analysis mechanism, we are particularly interested in analyzing the runtime behavior of bytecode instructions associated with critical operations that we might use in Axis that can be potentially involved in suspicious activities during the code execution.

Table 1: The most critical low-level instructions

OPCODE	INSTRUCTION	DESCRIPTION
0x55	SSTORE	Save word to storage
0xe2	SSTOREBYTES	Only referenced in pyethereum
0xf1	CALL	Message-call into an account
0xf2	CALLCODE	Message-call into this account with alternative account's code
0xf3	RETURN	Halt execution returning output data
0xf4	DELEGATECALL	Message-call into this account with an alternative account's code
0xfa	STATICCALL	Similar to CALL, but does not modify state
0xff	SELFDESTRUCT	Halt execution and register account for later deletion

Accordingly, we aim to introduce an efficient analysis system that identifies not only zero-day vulnerabilities and unseen attacks but also generates reliable test cases the identified bugs without human intervention in . Hence, we combined a hybrid approach based on "static call graph analysis", "dynamic execution", and symbolic execution in order to gain accurate results with maximum coverage.

Figure 7 represents the abstract architecture of our security approach in Axis.

Our approach works based on the following three main stages:

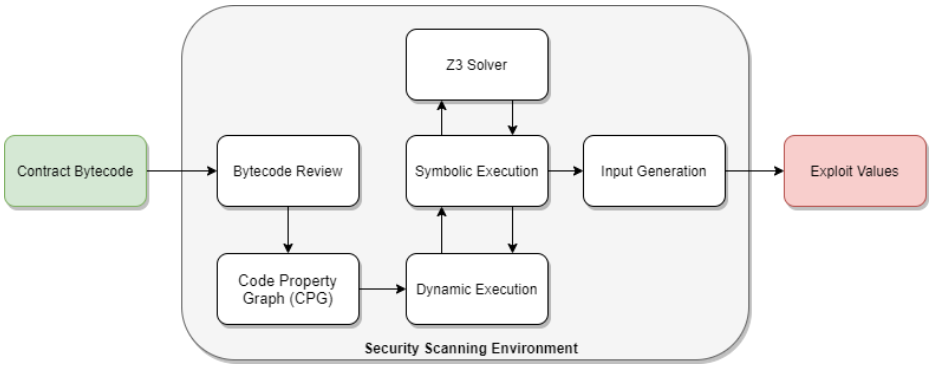

Fig. 7: The overview architecture of our security model

1. Code Property Graphs Analysis
2. Dynamic Execution
3. Symbolic Testing

3.3 Step 1: Call Graph Analysis

In order to interpret and identify potentially exploitable execution paths within the bytecode of smart contracts, we generate a graph model of the target smart contract in a first step. Our graph model is based on *Code Property Graphs (CPGs)* [27], which is an extensible and language-agnostic representation of program code designed for incremental and distributed code analysis. The CPGs are constructed based on the bytecode instructions to help us to distinguish critical instructions and the relevant execution paths. Note that a critical instruction in a generated CPGs can be managed to find data dependency paths between the variables. After finding these paths, we later execute them symbolically to reproduce their corresponding exploits. In order to generate the call graph we used "Porosity" [1], which is an open-source tool. However, the available source code contained many bugs. For example, it stopped the call graph at *STOP* and *REVERT* instructions and did not treat *STATICALL* as call instructions. Porosity only recognized *JUMPI* as jump instruction and thus ignored *JUMP* instructions.

Backward slicing. In order to generate concrete exploits, we also need to gain the correct entry point (we call it "source") so that a potential generated exploit can reach the critical instructions and perform an attack in the target contract successfully. To do so, EthFuzz performs a hybrid technique that comprises two steps, namely backward slicing" and non-exploitable path pruning (introduced in [3]), which is shown in Algorithm 1.

Algorithm 1: Performing backward-slicing to extract exploitable paths

```
   Input: Sensitive Instructions
   Result: Exploitable Paths
 1 initialization;
 2 SensitiveNodes ← FINSensitiveIntructionNode(SensitiveInstructions);
 3 foreach sn ∈ SensitiveNodes do
 4     ExploitableEV MPaths = ANALY ZECRITICALNODE(sn);
 5 end
 6 returnExploitableEV MPaths;
 7 Function ANALYZECRITICALNODE(vertex):
 8     ExploitableEV MPaths ← [] ;
 9     paths = BackwardSLC(sn);
10     foreach path ∈ paths do
11         if pathhasasource then
12             ExploitableEV MPaths ← path;
13         else
14             callPaths = ANALY ZECRITICALNODE(callV ertex);
15             ExploitableEV MPaths ← path + callPaths;
16         end
17     end
18     return ExploitableEV MPaths;
19 Function BackwardSLC(vetex):
20     IntraPaths ← [] ;
21     while vertex is not a source vertex is not a func. argument do
22         Incvertices = GETINCOMINGDDV ERTEX(vertex);
23         UnsanV ertics = FILTERSANNV ERTICS(Incvertices);
24         vertex ← unsanV ertixs;
25     end
26     IntraPaths = GETPATHSTO(vertex);
27     return IntraPaths;
```

The backward-slicing algorithm starts by investigating the nodes (presenting VM instructions) in the generated graph in order to draw critical instructions (line 2). For each node showing an instruction in the graph, EthFuzz explores its data dependency links in a backward way. *ANALYZECRITICALNODE* calls *BackSLC* in order to succeed all data dependency paths from an instruction node either to a source or a function argument. If the path drops at a function argument, *ANALYZECRITICALNODE* is called recursively over the points denoting the call-sites of that particular function. The function *BackSLC* then analyzes intra-procedural paths between sources and the critical nodes.

Pruning non-exploitable paths. To reduce the overhead caused by analyzing non-exploitable execution paths, we made an assumption. Suppose a detected path cannot reach a critical instruction in the contract under analysis. In that case, we consider the path as a non-exploitable path, which must be excluded from further analysis because it may cause overhead and false-positive results. Algorithm 2 represents the details of the pruning method for non-exploitable paths.

Algorithm 2: Pruning non-exploitable paths

```
Input: β: candidate path set, exposedCN: explosed critical node
Result: α: set of paths after pruning
foreach p ∈ Path do
    if isRelevant() == True then
        α.push(P);
    else
end
Function isExploitable(P):
    if Const.solve() == ∅ then
        return False;
    else
    if P.Succs ∩ exposedCN == ∅ then
        return False;
    else
    return True;
```

3.4 Step 2: Dynamic Execution

In our security approach, we also need to detect the presence of this type of protection in execution paths in order to reduce the potential of false-positives. To do so, we leverage dynamic execution to assess the exploitable target paths with actual runtime data. Thus, in the dynamic execution module, a special input or operation result, and a symbolic variable with a specific name (we call it "Taint Label") is added to the exploitable path. This symbolic variable is initially set by O and will be defined *TAINT-PC* (*PC* means program counter). We demonstrate this module in Figure 8.

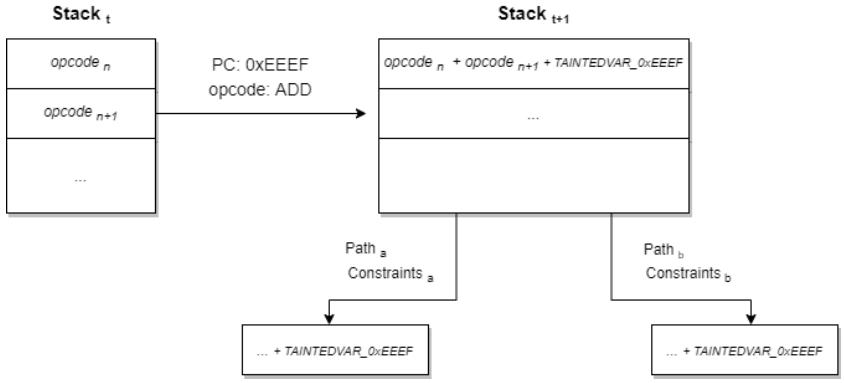

Fig. 8: Dynamic execution allows us to tracks all tainted inputs in the memory during the execution time.

The taint label covers to succeeding branches along with the potential dangerous data, engaging in computations but without modifying the results. Our taint tracking method is described as follows:

1. Performing data flow analysis based on the propagation rules that describe which operations can propagate taint message or lead to new taint messages during the execution time (e.g. *ADD*, *SUB*, *MUL*.).
2. At some specific program points, security-critical parameters or state variables are supposed to be tainted or not on-demand, according to whether they carry taint labels.
3. If security-critical data of a risky point was tainted, the source entry can also be detected based on the tainted label.

3.5 Step 3: Symbolic Testing & Test Case Generation

After identifying exploitable paths, EthFuzz starts to generate concrete exploits for the paths. To avoid path explosion issues, we perform this stage with the help of the collected actual and runtime data during the dynamic execution. Thus, our symbolic engine is not trapped in infinitive loops and infeasible paths. The exploit generation system operates based on a dynamic symbolic execution and Z3 SMT solver [7]. Consequently, we model the exploitable paths as a logical formula "F_{-Xpath}" so that their constraints are derived from the arguments of the extracted source "$F_{-Source}$" and critical instruction "F_{-Crins}" that represent which values after submitting to the target contract can lead to successful attacks.

As a result, the final formula is created as "$F_{-Xpath} \cap F_{-Source} \cap F_{-Crins}$" and will be sent to the Z3 SMT solver. The solver is responsible for executing the exploitable path symbolically and collects a set of path constraints to deliver the values (we call the values "payloads"). In the Z3 engine, we model the arguments of the call sites as "fixed-size" and "bit-vector" expressions. Moreover, we define the "variable-length" elements, such as the arguments by using the array expressions. The outcome of this modelling is actual practical exploits to trigger vulnerabilities inside target VM bytecode.

Note that EthFuzz generates exploits on a single path first, before seeking more extensive path sequences. Due to the relatively small size of smart contracts, EthFuzz explores path sequences up to length *10*, consisting of at most *8* state-changing paths and one last exploitable path. Listing 1.2 present a vulnerable contract and corresponding exploit generated by EthFuzz. As we stated before, we have implemented our symbolic execution engine based on the Z3 SMT solver.

4 Axis Mining Protocol

Modern blockchains have now been adequately in use for more than a decade. In addition to the growing desire for its key financial features (e.g., immovable, transparent, secure, and private), their potential to promote, establish, or enforce the agreement or performance of different transactions presents an outstanding innovative setting that will have a disruptive influence for many purposes. One disadvantage of main current blockchains (such as Bitcoin and Ethereum) is the

Listing 1.2: An example of a generated test case for a given smart contract bytecode

```
Location: from 22:27 to 22:46
2 numberTokens * COST_PER_TOKEN
3 ------------------------------
Transaction Sequence:
Tx #1:
Origin: 0xdeadbfefdeadbeefdeadbeefdeadbfefdeadbfee [ ATTACKER Address ]
Function: buy(uint256) [ d969094a ]
Data: 0xd969094a80100000000000000000000000000000000000000
00000000000000000000
Value: 0x0
```

moderation of block mining rate. With mining reward as a means in the control of the volume of values, there is tough competition in block mining. In order to avoid that extremely concurrent blocks are mined and proposed for commitment, they demand a "proof of work" by pushing the block miner to execute a high number of CPU cycles (or GPU, FPGA) by iterated hash computations before proposing her block. The concept of computational difficulty as proof of work is not new, it emerged first in the seminal work of Dwork and Noar and was later recommended to avoid Sybil attack by Aspnes et al. [2]. The proof of work provides rarity and uniqueness that help lessen the personal block mining rate.

However, the "Proof of Work" (PoW) at the time of writing this paper costs around $100,000,000,000$ CPU cycles (around 10 minutes on a current PC), and if one million miners contest at the same time, then the energy cost of mining one Bitcoin surpasses the value of the Bitcoin itself, this problem applies to Ethereum, Dogecoin, and a plenty of other blockchains as well. Moreover, this cost is anticipated to grow with the number of contending miners heading to a protocol whose energy cost can unquestionably not be sustainable in the near future.

In order to avoid these catastrophic outcomes, loads of alternative blockchains have been proposed and investigated (e.g., [30]), including choices of proofs of stake or various proofs of "something" (e.g., proof of Exercise, Proof-of-History (PoH) consensus method). Note that other endeavors to tackle the enormous waste generated by the Proof of Work as implemented in Bitcoin have been directing at turning them into work that is actually useful. For instance, solving functional problems have been studied in [21], stretched to present PoWs that are based on 3SUM, Orthogonal Vectors, and All-Pairs Shortest Path puzzles. They can also be utilized to provide for much faster verification and zero-knowledge PoWs. However, it is still unclear how these approaches can produce incentives for the high amount of transactions and how they guarantee Liveness (and in particular, independence from starvation in even no one volunteers to resolve a practical puzzle).

Accordingly, In Axis architecture, we concentrate on a specific unsustainable energy cost and propose a cryptocurrency protocol moderated through a super optimum mining protocol in such a way that we will eliminate the prohibitive energy cost of the proof of work while keeping the blockchain mining input rate between reasonable limits. Moreover, this mining protocol will make our blockchain compatible with the most popular blockchains such as Ethereum. Those boundaries can be arbitrarily harmonized *a priori*, while the proof-of-work CPU parameter must be constantly updated. Furthermore, in our mining protocol, users can mine the blocks even with lightweight processors used in mobile phones and smart devices, which enables mobile developers to expand and develop numerous projects on top of Axis.

4.1 Fairness

Axis's validation process warrants a tunable and defined average number of concurrent mining whatever the size of the population in competition, similar to an election. In addition to public verifiability, our scheme implements fairness and liveness so that the unique chain expands and the probability of selecting each party is independent to its relative computational power, which results in fairness in our network.

4.2 Permissionless Model & Fork Protection

Axis's model is "permissionless", which means there is no explicit membership protocol; anyone can join the system without any discrimination. Axis model also extends the protection property so that blocks are ultimately committed and cannot be separated. Finally, Axis scheme decreases the risk of Forks (il.e., divergent chains of blocks that require to be eventually dropped) and presents tunable scalability for the number of users and the pace of block generation.

4.3 Innovative Nonce-less Block Generation

Our mining protocol employs a block generation scheme that does not use a nonce. The innovation of Axis protocol comes from a call field that controls the rate of creation of transaction blocks, in addition to empty blocks. The methodology of the proof serves a distributed election via a collusion scheme that has an interest on its own. The method guarantees a tunable and limited average number of concurrent mining, whatever the size of the population in competition for block generation, thus offering full scalability. Note that our protocol for implicit empty blocks executes the scheme fully distributed, tunable and scalable. As the scheme also lessens the chances of forks, it further overcomes energy waste.

5 Axis's Mining Protocol Formalization

5.1 Parameters and format

Axis mining mechanism is based on the following parameters:

- an integer k;
- a vector of increasing probabilities of length $k + 1$: $P_0, P_1, \ldots, P_k$, with $P_k = 1$;
- an upper bound N on the maximum admissible number of contending blocks, that can be set as large as possible.

Typical values for these parameters would be $k = 8$, $N = 2^{32}$. The Axis protocol consists in authorizing the mining of an average number of order $N^{1/k}$ out of $n \leq N$ contending blocks.

Each block in the Axis protocol will bring some values retrieved by a hash function. We express h the length of the hash value fields in the block. A standard value is $h = 256$ (32 bytes).

Our mining protocol relies on two classes of blocks:

- *Empty* Blocks: They do not hold any transactions.
- *Regular* Blocks: They include transactions.

The format of a regular block is comparable to the Ethereum block format (for the sake of compatibility). However, the uniqueness of our format is that a regular blockchain does not carry a *nonce* field, but instead, it contains a *call* field. This call field delivers the demanded criteria for the next hash value field in the chain. The following table presents an abstract format placing the most significant fields in the block:

1	previous block's hash
2	date
3	transactions' list
...	...
4	call value
5	the block hash

The empty block resembles similar but without a transaction list.

5.2 Mining protocol

To explain our presentation, we will initially consider empty slots mined by a fundamental entity and on a limited block server.

A block in the Axis ecosystem is mined only if its hash value is smaller than the call value of the previous block. The call field of a regular block perpetually has the same value, which is $\lfloor 2^{h+1} P_0 \rfloor - 1$. If no regular block is mined, the primary entity mines an empty block after a specific lapse of time. Moreover, unlike regular block hash values, an empty block hash value does not expect to

be smaller than the call value of the previous block. The call value of the first empty block must have value $\lfloor 2^{h+1}P_1 \rfloor - 1$.

When a new regular block is mined, the primary entity mines empty block with successive call values $\lfloor 2^{h+1}P_2 \rfloor - 1, \ldots, \lfloor 2^{h+1}P_k \rfloor - 1$. After k mined empty blocks the call values sequence restarts.

In this regard, there are **three facts** points:

- If there is no nonce field in the regular block, no proof of work computation is required.
- Instead, the blocks are authorized via the call values (they have the effect of regulating the mining rate)
- The non-authorized blocks do not require to be submitted, which also restricts the traffic generated by the block mining.

Liveness. If a call fails, *i.e.* no regular block is mined, then a new call will be made through a new empty block. Since $P_k = 1$, the last call value is the max-value $2^{h+1} - 1$. Consequently, any competition of block mining will result in at least one block mined.

Figure 9 displays an instance of block sequences in the Axis blockchain. The chain is enforced by the hash values $Y_0, Y_1,$ The value of a regular block hash should always be smaller than the call value of the previous block; regardless it is empty or regular. Empty blocks do not need to reflect this rule.

Fig. 9: A sequence of blocks, empty blocks are grayed/yellow.

6 Performance Analysis

In this part, we introduce our analysis of our selection mechanism, our protocol's performance, and its scalability. The election aims to select leaders among $n > 0$ users by proceeding through several rounds. Note that practicing rounds is also commonly accepted that the Blockchain model includes some degree of synchrony assumptions. In a network with infinite delay, the adversary can heighten its power just by making the non-faulty parties provoke higher delays (e.g., [**?**]). We will reveal that these assumptions are irrelevant to our scheme.

Informally, the rounds of our election are made through coin tossing. We believe that each competitor has an Axis token whose head probability is p and tail probability $q = 1 - p$. We want to advance to an election among $n \geq 1$ competitors. In the beginning, all the n competitors transmit. If this results in a collision, then each competitor tosses the coin; only those who get ahead contest for the second round, we call them the survivors. Suppose the second-round results in a collision. In that case, a second coin-tossing

happens. A subset of survivors contest, and the protocol continues until none survive. In this case, we take as a result the last non-empty set of survivors. Normally there is a further procedure to minimize the survivor population to a single using a collision algorithm. In our case, this is not forced as long as the subset of survivor is of reasonable size: we will prove using analytic combinatorics that the average size of the last non-empty survivor set is $\frac{q}{p\log(1/p)}$ with some fluctuations of small amplitude. Note that this value is limited and sovereign of the initial number of contenders. This average size can also be tuned to be small or large by tuning the parameter p. This can be done by negotiating with the average number of rounds, which tends to $\log_{1/p} n$ when $n \to \infty$.

Suppose we needed to copy this process for the call values in the empty block. In that case, we will require a sequence $(P_0, P_1, \ldots)$ such that $P_A = p^A$ describes an exponential descending staircase; however, this will compose the case that after the first call one would have already n mined blocks. But, we can use the Axis protocol to build the ascending exponential staircase effect when there is a large yet fixed limit N to the number of simultaneous contenders. In this case, we presume that $P_0 = \frac{1}{N}$ and that $P_A = \frac{1}{N} p^{-A}$ for $I \leq k$. In order to have $P_k = 1$ one must have $p = \frac{1}{N^{1/k}}$. If $N = 2^{32}$ and $k = 8$ we will have $p = \frac{1}{16}$ and the call sequence will be:

call value rank $0 \leq I \leq k = 8$	Probability sequence $P_I = N^{\frac{I}{k}-1}$	call value $\lfloor 2^{h+1} . P_I \rfloor - 1$
0 (initial)	2^{-32}	$2^{224} - 1$
1	2^{-28}	$2^{228} - 1$
2	2^{-24}	$2^{232} - 1$
3	2^{-20}	$2^{236} - 1$
4	2^{-16}	$2^{240} - 1$
5	2^{-12}	$2^{244} - 1$
6	2^{-8}	$2^{248} - 1$
7	2^{-4}	$2^{232} - 1$
8	1	$2^{256} - 1$

We express $\mathbf{M}_n$ the average number of regular mined blocks when n blocks compete with a regular block. Note that this paper's essential yet regular hypothesis is that succeeding calls of a hashing function are autonomous. Of course *stricto sensu* this is not true since hash value enterprises are deterministic calculations. However, the better succeeding hash values imitate independent random variables, the better is a hash function. Indeed this argument is the basis for accepting the resilience of the Axis blockchain (made of successive hash computations) against attacks.

Let assume that $\mathbf{M}_n^A$ indicates the number of blocks mined after the Ith empty block under the condition that all the previous empty blocks resulted into no regular block mined. We have $\mathbf{M}_n = \mathbf{M}_n^0$ and since $P_k = 1$: $\mathbf{M}_n^k = n$.

Since the fact that the successive calls to hash values are assumed independent, the number of blocks called by the Ith empty block is a binomial random variable $B(n, P_A)$ of probability P_A. For $I < k$ when the binomial

variable is $B(n, P_A) = 0$ which occurs with probability $(1 - P_A)^n$, no block is mined after the lth block and $\mathbf{M}_n^A = \mathbf{M}_n^{A-1}$. Hence, for all integers m, $m > 0$:

$$P(\mathbf{M}_n^A = m) = \binom{n}{m} P_A^m (1 - P_A)^{n-m} + (1 - P_A)^n P(\mathbf{M}_n^{A+1} = m)$$

and

$$P(\mathbf{M}_n^A = 0) = \delta(n). \tag{1}$$

where $\delta(n) = 1$ if $n = 0$ and zero otherwise.

Let $M_n^A(u) = E[u^{\mathbf{M}'_n}]$.

Lemma 1. *We have for $I < k$*

$$M_n^A(u) = (1 + P_A(u - 1))^n - (1 - P_A)^n + (1 - P_A)^n M_n^{A+1}(u) \tag{2}$$

and $M_n^k(u) = u^n$.

This is a direct application of Equation (1).

Lemma 2. *We have the identity*

$$E[\mathbf{M}_n] = nP_0 + \sum_{A=1}^{A=k} nP_A \prod_{j<A} (1 - P_j)^n \tag{3}$$

This is a direct application of previous lemma by using $E[u^{\mathbf{M}'_n}] = \frac{\partial}{\partial u} M_n^A(1)$.

Theorem 1. *For all $n \le N$, we have the estimate*

$$E[\mathbf{M}_n] = O(N^{1/k}).$$

Remark the optimal value is $k = O(\log N)$ but lower values are also interesting. From Equation 3 we get the inequalities

$$E[\mathbf{M}_n] \le nP_0 + \sum_{A=1}^{k} nP_A (1 - P_{A-1})^n$$
$$\le nP_0 + \sum_{1 \le A \le k} nP_A \exp(-P_{A-1} n).$$

$$\sum_{1 \le A \le k} nP_A \exp(-P_{A-1} n) = \sum_{1 \le A \le k} \frac{n}{N} p^A \exp(-p^{A-1} n/N)$$

This quantity is smaller than $f(n/N)$ with $f(x) = \frac{1}{p} \sum_{A \in Z} g(p^A x)$ with $g(x) = xe^{-x}$. The function $f(e^x)$ is periodic of period $\log p$ and therefore is bounded. Because of $p = N^{-1/k}$, $n/N \le 1$ and $n \le N$, we obtain the result.

Theorem 2. *We have the more precise estimate*

$$E[\mathbf{M}_n] \leq \frac{n}{N} + AN^{1/k}. \tag{4}$$

with $A = \frac{1}{k\in\mathbb{Z}\log(1/p)} \sum |\Gamma(1 + 2ik\pi/\log p)|$, *p denoting* $N^{-1/k}$ *and where* $\Gamma(.)$ *is the Euler "Gamma" function.*

We can give an accurate estimate of the maximum value of function $f(x)$. The function $f(e^x)$ is periodic of period $\log p$ and therefore has a Fourier decomposition:

$$f(e^x) = \sum_{k\in} g_k e^{2ik\pi x/\log p} \tag{5}$$

where g_k is the Fourier coefficient.
We have:

$$\begin{aligned} g_k &= \int_{\log p}^{0} f(e^x) e^{2ik\pi/\log p} dx \\ &= \frac{1}{\log(1/p)} \int_{0}^{\infty} g(x)^{2ik\pi/\log p - 1} dx \\ &= \frac{1}{\log(1/p)} \Gamma(1 + 2ik\pi/\log p) \end{aligned}$$

and then use the upper bound $|f(e^x)| \leq \sum_{k\in Z} |g_k|$.
In passing we obtain the mean value of $f(e^x)$ to be equal to $g_0 = \frac{1}{\log(1/p)}$.
Figure 10 represents the quantity $E[\mathbf{M}_n]$ versus n for various parameters. The sequence of bumps reflects the periodic fluctuations as function of $\log n$ analyzed in the upper bound.
Figure 11 shows the exact amounts obtained by simulation, each point being simulated 1,000 times. We can yield a relative expression of the distribution of $\mathbf{M}_n$.

Lemma 3. *For all complex number u:*

$$\begin{aligned} E[u^{\mathbf{M}_n}] = (1 + P_0(u - 1))^n - (1 - P_0)^n \\ + \sum_{0<A<k\, j<A} \prod (1 - P_j)^n \\ \times ((1 + P_A(u - 1))^n - (1 - P_A)^n) \\ + u^n \prod_{j<k} (1 - P_j)^n. \end{aligned}$$

By application of Lemma 1.
Figure 12 displays the various shape of the sequences $P(\mathbf{M}_n > m)$ versus integer m for various parameters. Notice that the plot sequence of descending bumps arises from the mixture of the separate binomial distributions. We can also estimate the exponential rear distribution of $\mathbf{M}_n$ by the following theorem regarding large deviations.

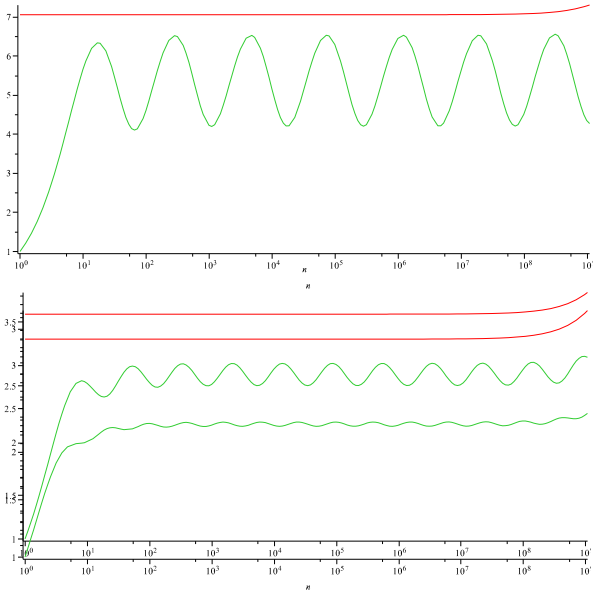

Fig. 10: The quantity $E[\mathbf{M}_n]$ and the upper bound given by (4) in red versus n for $N = 2^{32}$, (left) for $k = 8$, (middle) $k = 12$, (right) $k = 16$.

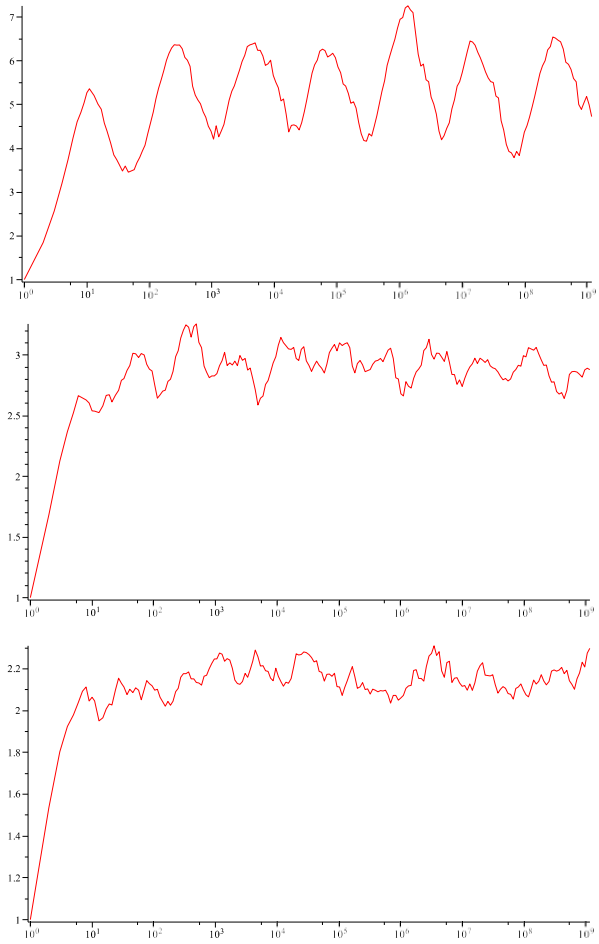

Fig. 11: The same as in figure 10 $E[\mathbf{M}_n]$ with $N = 2^{32}$ but obtained with 1,000 simulations per point, (left) for $k = 8$, (middle) $k = 12$, (right) $k = 16$.

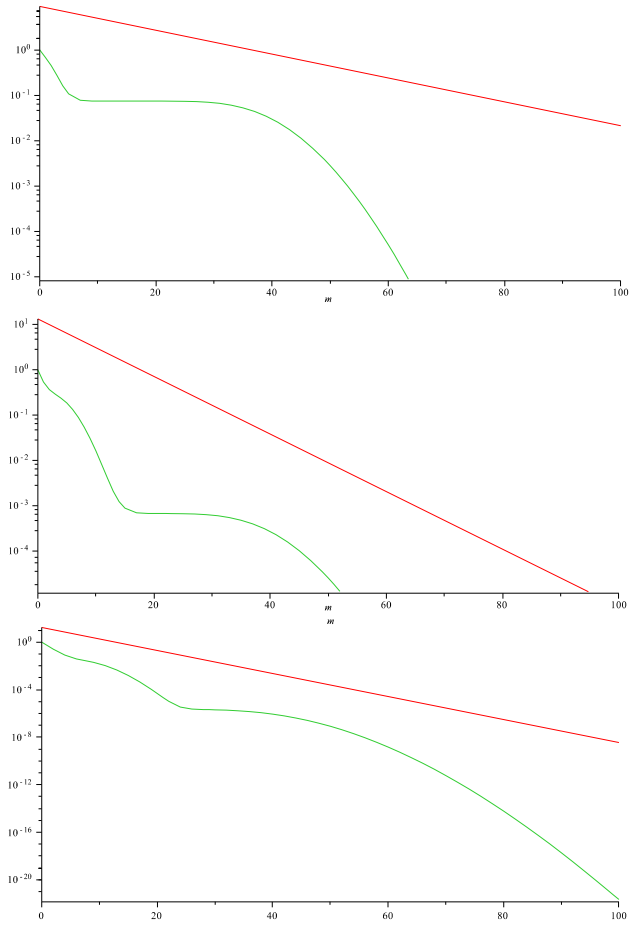

Fig. 12: Computed values of $P(\mathbf{M}_n > m)$ with upper bound given by (6) in red versus m for $n = 10,000$, with $N = 2^{32}$, (left) for $k = 8$, (middle) $k = 12$, (right) $k = 16$.

Theorem 3. *Let $p = N^{-1/k}$. For all integer $m > 0$, we have the estimate*

$$P(\mathbf{M}_n > m) \leq \frac{k + e^p}{(1+p)^m} \tag{6}$$

We have $P_A = p^{k-A}$ and

$$E[u^{\mathbf{M}n}] \leq (1 + p^k(u-1))^n - (1-p^k)^n + \sum_{0<Æ<k} \left(1 + p^{Æ}(u-1)\right)^n - (1-p^{Æ})^n \quad (1-p^{Æ+1})^n + u^n(1-p)^n$$

Let $u = 1 + p$, we have $1 + (u - 1)p^A = 1 + p^{A+1}$ and therefore $(1 + (u - 1)p^A)(1 - p^{A+1}) = 1 - p^{2(A+1)} \leq 1$. Thus $E[u^{\mathbf{M}_n}] \leq (1 + p^{k+1})^n + k$. Since $(1 + p^{k+1})^n \leq \exp(np^{k+1}) = \exp(pn/N)$ and $n < N$ we have $\exp(np/N) \leq \exp(N^{-1/k})$. We conclude the proof with the observation that for all integer m: $u^m P(\mathbf{M}_n > m) \leq E[u^{\mathbf{M}_n}]$.

Figure 13 illustrates the histograms of 1,000 simulations of $\mathbf{M}_n$ versus n for $N = 2^{32}$ and $k = 8, 12, 16$. The color of the point (n, m) indicates the number of times the event $\mathbf{M}_n = m$ has been obtained during the simulation with the color code: black when more than 64 times, red when between 16 and 64, blue when between 4 and 16, green when between 1 and 4.

7 Decentralized Empty Block Mining in Axis

To gain the empty block production fully distributed, we continue the Axis mining scheme in two steps (with the introduction of two features): (A) Time moderated empty blocks mining; (B) implicit empty blocks mining.

7.1 Time moderated empty block mining

Here all entities can mine empty blocks. To evade unfairness between empty miners, a new empty block will be supplied only if its date is beyond a minimal gap time between the last full and empty block time. Correlating with other existing schemes, the Minimal Gap Time (MGT) can be set to one minute (e.g., in Bitcoin, this is the Block Time, the average Time between blocks, and the difficulty of the PoW is adjusted to make the Time about 10 minutes; while the block time for Ethereum is set to between 14 and 15 seconds). Note that, in this case, we still require to have at least one entity mining empty blocks to secure liveness. Consequently, for this first variation, we find that possibly different (and at least one) peer-nodes generate empty blocks and therefore play the role of distributed timestamping and block servers.

In the Axis ecosystem, time can be achieved using Unix universal time (POSIX), a broadly used timestamping system in Unix-like operating systems that are regularly used for standard Blockchain systems. However, one

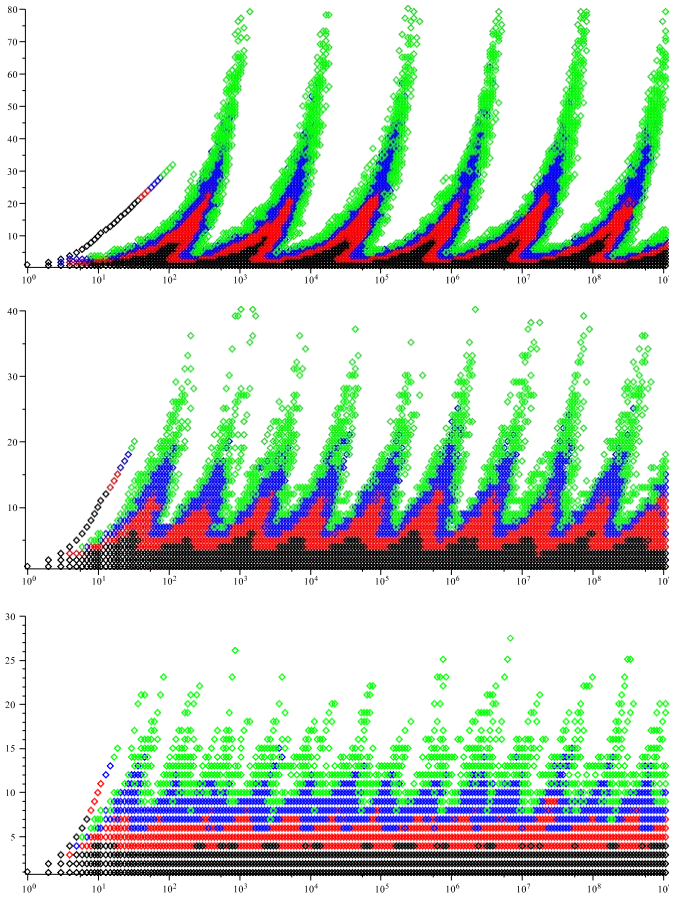

Fig. 13: various values of $\mathbf{M}_n$ versus n simulated 1000 times black values appeared more than 64 times, red more than 16 times, blue more than 4 times, green more than once, with $N = 2^{32}$, (left) for $k = 8$, (middle) $k = 12$, (right) $k = 16$.

problem is that the date given by the clock is not mandatorily sound (e.g., Unix time is not a precise depiction of UTC, and leap seconds are not considered for). But, note we are only interested in the time interval in this part. One possibility is that an extra timestamp is added to the block to reflect the local time when the timestamping server held the block. Considering the timestamping server's time is local, it can only be appended at the reception and will be excluded from the hash value calculation of the block. An empty block obtained before the termination of the minimum time gap with the local time of the timestamping server will be delayed. An empty block with a timestamp that does not show the MGT offset with the original timestamp of a full block will be discarded.
The date field of the empty blocks must be excluded from their hash value computation. Contrarily, full blocks with different hash values could be called by various empty blocks on different peers because the latter dispenses different hash values due to separate time stamps. Therefore, the sequence of hash values of the empty blocks mined after a full block will follow a deterministic sequence. If we disregard the clock drifts, the use of the time stamp does not alter the performance analysis of the scheme as two consecutive calls to the hash function produce autonomous hash values.

7.2 Implicit empty block mining

We can now extend the previous scheme because the sequence of hash values of the empty blocks mined after an entire block will follow a deterministic sequence and thus make them "implicit". In addition, and finally, we remove the requirement of any particular or centralized entity entirely and make the scheme fully distributed: one does not need a unique or central entity to initiate empty blocks since empty blocks will now become implicit, i.e., no empty blocks will be mined. The modification of the protocol is as follows: (i) Upon reception of a whole block within less than one MGT after the mining of the last whole block, the block server discards it; (ii) Upon reception of an entire block within less than l MGT ($1 \leq l \leq k$) after the mining of the last whole block, with either new block's hash value, is more prominent than $2^h p_{A-1}$ or the previous block hash value is not the hash value of the last entire block, a block server discards it. Note that another advantage of this scheme is that it also minimizes the risk of creating forks as entire blocks are discarded earlier.

7.3 Performance analysis of the implicit empty block scheme

The main difference with the previous scheme analysis is that the hash value of the blocks after each empty block is no longer independent. The hash value of an entire block candidate is no longer possibly related to a new empty block. Indeed the previous block hash value field is no longer the hash value of the previous empty block since it no longer exists, but the hash value of the last whole block. Keeping the previous notations, we can state our main theorems and lemmas.

Lemma 4. *For $m > 0$ we have the expression*

$$P(\mathbf{M}_n = m) = \binom{n}{m} P_0^m (1 - P_0)^{n-m} + \sum_{A=1}^{k-1} \binom{n}{m} (P_A - P_{A-1})^m (1 - P_A)^{n-m} + \delta(m - n)(1 - P_{k-1})^n$$

We call *slot* the period of length MGT. Let suppose that n entire blocks compete. Let $0 < l < k$, the chance that a block has a hash value contained between $2^h P_{A-1}$ and $2^h P_A$ is $P_A - P_{A-1}$. Such a block will be mined on the $l+1$-th slots after the last full block if and only if all the other full blocks in the competition have their hash values greater than or equal to $2^h P_A$. Those which have a hash value greater than $2^h P_A$ will not be mined. The hash values of the blocks being considered sovereign justify the binomial expression. The case $l = 0$ corresponds to blocks with a value smaller than or equal to $2^h P_0$. In the case $l = k$ we have $P_k = 1$ and occur when all the blocks have hash values higher than or equal to $2^h P_{k-1}$

Lemma 5. *We have the expression:*

$$E[\mathbf{M}_n] = nP_0 + \sum_{A=1}^{k-1} n(P_A - P_{A-1})(1 - P_{A-1})^{n-1}. \tag{7}$$

This is a direct expression of the previous lemma. We have $E[\mathbf{M}_n] = \frac{\partial}{\partial u} E[u^{\mathbf{M}_n}]|_{u=1}$. Indeed for u complex: $\sum_m u^m \binom{n}{m} (P_A - P_{A-1})^m (1 - P_{A-1})^{n-m} = ((P_A - P_{A-1})u + 1 - P_{A-1})^n$ and the derivative of the right hand side with respect to variable u is equal to $n(P_A - P_{A-1})((P_A - P_{A-1})u + 1 - P_A)^{n-1}$. We notice that when $l = k$ we have $n(P_A - P_{A-1})(1 - P_{A-1})^{n-1} = n(1 - P_{k-1})^n$ since $P_k = 1$. The performance of the implicit empty blocks approach is analogous to the performance of the explicit empty blocks. In fact, we realized that the divisions of the medium number of mined blocks oscillate and cross each other when we match both strategies (see Figure 14).

Theorem 4. *For $n \leq N$ we have $E[\mathbf{M}_n] = O(N^{1/k})$.*

We have $E[\mathbf{M}_n] - P_0 n = \sum_{A=1}^{k} n(P_A - P_{A-1})(1 - P_A)^{n-1}$. Since $P_A = p^{k-A}$ and $p^k = 1/N$ we get

$$\begin{aligned} E[\mathbf{M}_n] - \frac{n}{N} &= \sum_{A=1}^{k-1} n(1/p - 1)P_A(1 - P_{A-1})^{n-1} \\ &= \sum_{A=1}^{k-1} n(1/p - 1)P_A \frac{(1 - P_{A-1})^n}{(1 - P_{A-1})} \\ &\leq \frac{1}{p} \sum_{A=1} nP_A(1 - P_{A-1})^n \end{aligned}$$

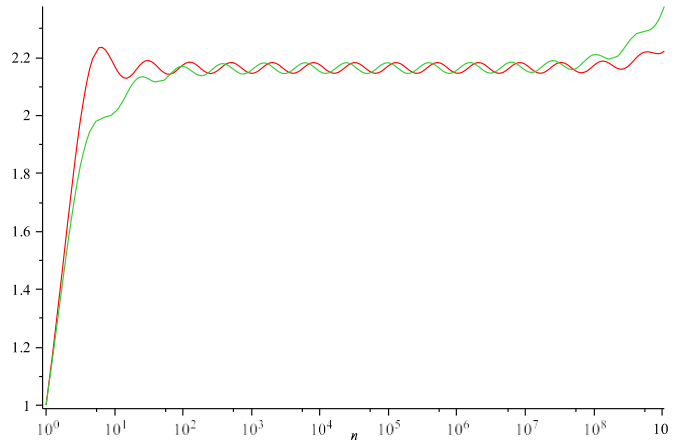

Fig. 14: Average number $E[\mathbf{M}_n]$ of mined blocks with explicit (green) and implicit (red) empty blocks mining versus n for $k = 16$.

since $(1 - P_{A-1})^{-1} \leq (1 - p)^{-1}$.
The last right hand term is already proven to be $O(N^{1/k})$ in the proof of Theorem 1. Using the same reasoning, we obtain a more precise estimate via the use of the "Gamma" function: $E[\mathbf{M}_n] \leq \frac{n}{N} + BN^{1/k}$ with $B = \frac{1}{p \log(1/p)} \sum_{k \in Z} \Gamma\left(1 + \frac{2ik\pi}{\log p}\right)$.

8 Conclusion

In this whitepaper, we introduced the Axis blockchain as a practical, fast, secure, and flexible blockchain technology for robust DApp developments. Our approach considers the complexities of real-world decentralized applications and effectively addresses them. Moreover, our evaluation results prove our blockchain concept's usefulness, performance, energy efficiency, and adaptability. Note that the Axis blockchain is under constant development, and we will release its source code publicly in the near future.

References

1. Github - comaeio/porosity: *unmaintained* decompiler and security analysis tool for blockchain-based ethereum smart-contracts. https://github.com/comaeio/porosity. (Accessed on 05/07/2020).
2. Kotaiba Alachkar and Dirk Gaastra. Blockchain-based sybil attack mitigation: A case study of the i2p network. *August*, 22:1–13, 2018.
3. Mohammadreza Ashouri. Kaizen: a scalable concolic fuzzing tool for scala. In *Proceedings of the 11th ACM SIGPLAN International Symposium on Scala*, pages 25–32, 2020.

4. Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust*, pages 164–186. Springer, 2017.
5. R Bhuvana and PS Aithal. Blockchain based service: A case study on ibm blockchain services & hyperledger fabric. *International Journal of Case Studies in Business, IT and Education (IJCSBE)*, 4(1):94–102, 2020.
6. Andrew D Birrell and Bruce Jay Nelson. Implementing remote procedure calls. In *Proceedings of the ninth ACM symposium on Operating systems principles*, page 3, 1983.
7. Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
8. Paul F Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122–127, 1982.
9. Seyoung Huh, Sangrae Cho, and Soohyung Kim. Managing iot devices using blockchain platform. In *2017 19th international conference on advanced communication technology (ICACT)*, pages 464–467. IEEE, 2017.
10. Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: Analyzing safety of smart contracts. In *NDSS*, pages 1–12, 2018.
11. Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004.
12. Angwei Law. *Smart contracts and their application in supply chain management*. PhD thesis, Massachusetts Institute of Technology, 2017.
13. Wei-Meng Lee. Using the web3. js apis. In *Beginning Ethereum Smart Contracts Programming*, pages 169–198. Springer, 2019.
14. Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *Future Generation Computer Systems*, 107:841–853, 2020.
15. Antonio López Vivar, Alberto Turégano Castedo, Ana Lucila Sandoval Orozco, and Luis Javier García Villalba. Smart contracts: A review of security threats alongside an analysis of existing solutions. *Entropy*, 22(2):203, 2020.
16. Dinh C Nguyen, Pubudu N Pathirana, Ming Ding, and Aruna Seneviratne. Blockchain for 5g and beyond networks: A state of the art survey. *arXiv preprint arXiv:1912.05062*, 2019.
17. Santiago Palladino. The parity wallet hack explained. *July-2017.[Online]. Available: https://blog.zeppelin.solutions*, 2017.
18. Daniel Perez and Benjamin Livshits. Smart contract vulnerabilities: Does anyone care? *arXiv preprint arXiv:1902.06710*, 2019.
19. Haseeb Qureshi. A hacker stole 31 m of ether—how it happened, and what it means for ethereum. *Freecodecamp. org, Jul 20, 2017*, 2017.
20. Mayank Raikwar, Danilo Gligoroski, and Katina Kralevska. Sok of used cryptography in blockchain. *IEEE Access*, 7:148550–148575, 2019.
21. Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. Survey of consensus protocols on blockchain applications. In *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–5. IEEE, 2017.
22. Ali Ashkanan Sa'ed Abed, Wathiq Mansoor, and Amjad Gawanmeh. Enhanced sat solvers based hashing method for bitcoin mining. In *17th International Conference on Information Technology–New Generations (ITNG 2020)*, volume 1134, page 191. Springer Nature, 2020.

23. E Gün Sirer. Thoughts on the dao hack. *Hacking*, 17, 2016.
24. Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18:2, 1996.
25. Tsankov and Dan. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82. ACM, 2018.
26. 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.
27. Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*, pages 590–604. IEEE, 2014.
28. Jing Zeng, Chun Zuo, Fengjun Zhang, Chunxiao Li, and Longshuai Zheng. A solution to digital image copyright registration based on consortium blockchain. In *Chinese Conference on Image and Graphics Technologies*, pages 228–237. Springer, 2018.
29. Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.
30. Yijun Zou, Ting Meng, Peng Zhang, Wenzhen Zhang, and Huiyang Li. Focus on blockchain: A comprehensive survey on academic and application. *IEEE Access*, 8:187182–187201, 2020.