International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Systematic study on vulnerabilities, countermeasures and security analysis tools for cryptocurrencies and post quantum cryptographic approach to design quantum resistant blockchains

Krishnamoorthy. V * and Marikkannan.M,

Department of Computer Science and Engineering, Government College of Engineering, Erode.

International Journal of Science and Research Archive, 2024, 13(02), 1416–1433

Publication history: Received on 14 October 2024; revised on 21 November 2024; accepted on 23 November 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.13.2.2284

Abstract

Blockchain and other Distributed Ledger Technologies (DLTs) have evolved significantly in the last years and their use has been suggested for numerous applications due to their ability to provide transparency, redundancy and accountability. Public key cryptography and hash functions provide security to blockchains, smart contracts and cryptocurrencies. Fast progress in Quantum computing presents significant threats to blockchain systems as it has leveraged possibility of attacking cryptosecurity systems in near future forcing re-design of blockchains to withstand quantum attacks and vulnerabilities. This article studies different types of vulnerabilities to blockchains, existing countermeasures for attacks and investigates causes and preventive mechanisms to make quantum resistant or quantum proof blockchains. Thus, this article seeks to provide a broad view on security analysis tools, quantum-based cryptosystems, signature schemes and algorithms on Post-Quantum Blockchain (PQB) security to future blockchain researchers and developers.

Keywords: Post Quantum Cryptography; Vulnerabilities of Blockchains; Quantum attacks; Security tools for blockchain; Preventive methods of vulnerabilities in blockchain, Post Quantum Signing Algorithms.

1. Introduction

Blockchain is a technology that was born with the cryptocurrency Bitcoin [1] and that is able to provide secure communications, data privacy, resilience and transparency [2]. A blockchain acts as a distributed ledger based on a chain of data blocks linked by hashes that allow for sharing information among peers that do not necessarily trust each other, thus providing a solution for the double-spending problem [3][5]. Such features have popularized blockchain in the last years and it has already been suggested as a key technology for different applications related to smart health [6], measuring systems [7], logistics [8], [9], e-voting [10] or smart factories [11], [12].

Blockchain users interact securely with the blockchain by leveraging public-key/asymmetric cryptography, which is essential for authenticating transactions. Hash functions are also key in a blockchain, since they allow for generating digital signatures and for linking the blocks of a blockchain. The problem is that both public-key cryptosystems and hash functions are threatened by the evolution of quantum computers.

In the case of public-key cryptosystems, secure transaction data may be recovered fast by future quantum computing attacks. Such attacks impact the most popular public-key algorithms, including RSA (Rivest, Shamir, Adleman) [13], ECDSA (Elliptic Curve Digital Signature Algorithm) [14], [15], ECDH (Elliptic Curve Diffe-Hellman) [16] or DSA (Digital Signature Algorithm) [17], which can be broken in polynomial-time with Shor's algorithm [18] on a sufficiently

* Corresponding author: Krishnamoorthy. V

Copyright © 2024 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution Liscense 4.0.

powerful quantum computer. Moreover, quantum computers can make use of Grover's algorithm [19] to accelerate the generation of hashes, which enables recreating the entire blockchain. Furthermore, Grover's algorithm may be adapted to detect hash collisions, which can be used to replace blocks of a blockchain while preserving its integrity.

This article categorizes the types of block chain vulnerabilities in to three segments as Solidity Programming, Blockchain design Vulnerabilities and Quantum Computing vulnerabilities. Furthermore, it summarizes the causes, impacts and recommended preventive counter measures for non-quantum attacks and identifies potential area for research focus especially in transition to post quantum cryptosystems for block chains.

2. Vulnerabilities in cryptocurrencies

Based on previous works [20,21,22,23,24], here we have summarized several causes for cryptocurrency vulnerabilities.

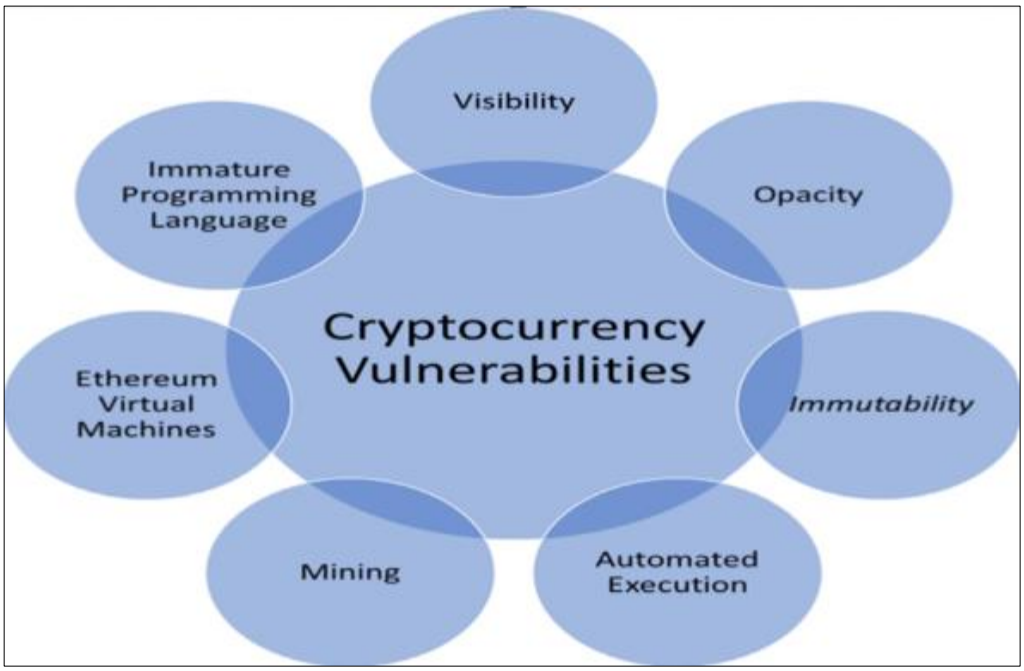

Figure 1 Cryptocurrency Vulnerabilities

2.1. Visibility

Blockchain records all of its validated historical transactions in world state, which is visible to the whole network. This visibility may cause some problems when the smart-contract owner wants to keep it from the public due to privacy concerns. Prior studies have shown that inspecting accumulated transactions' statistic characteristics and leveraging analysis graph structures could reveal valuable

2.2. Opacity

The opacity of smart contracts poses another concern. Live contracts need rigid inspection and source-code audits since they manage millions of USD. However, 77% of running smart contracts are opaque and hold USD 3 billion [28].

2.3. Immutability

Smart contracts cannot be modified once deployed on Ethereum. This immutability perpetuates problematic contracts on the chain. Workarounds such as updating contracts or addresses at the user side could solve the problem to a certain degree, however, they introduce other security concerns.

2.4. Automated Execution

A Control Flow Transfer (CFT) between two instructions A and B means that after A's execution, B is executed immediately. When smart contracts interact with each other, those CFTs are not always under control. In the

uncontrolled- function call-permission cases, the self-elevation of a malicious attacker could happen by triggering some high-privilege functions [29].

2.5. Mining

The order and sequence of processing the transactions to be processed depends on the miner's choice. The deterministic result of a single transaction's execution no longer exists when multiple transactions are involved. Their execution order may vary based on the miner's choice. This uncertainty severely affects order-sensitive transactions. The final result of a series of transactions might reach vastly different outcomes.

2.6. EVM

EVM structure provides a stand-alone, isolated running-time environment (sandbox) for smart contracts. EVM utilizes an oracle to handle the incoming real-world inputs. An oracle is a concept that bridges blockchain and the real world and acts as APIs on chain, which can be accessed by the smart contracts to receive feeds from the real world [28]. Centralized oracles introduce several attack vectors and issues to smart contract such as integrity, accuracy, and consistency [30]. However, the decentralized designs of oracle such as Chain link [31] are trying to solve this problem.

2.7. Immature Programming Language

Solidity is an evolving programming language and its widely known vulnerability is the re-entrancy problem, which is caused by the fallback mechanism in Solidity. Mishandled exceptions could be a severe problem for smart contracts too.

3. Blockchain vulnerabilities classification and related works

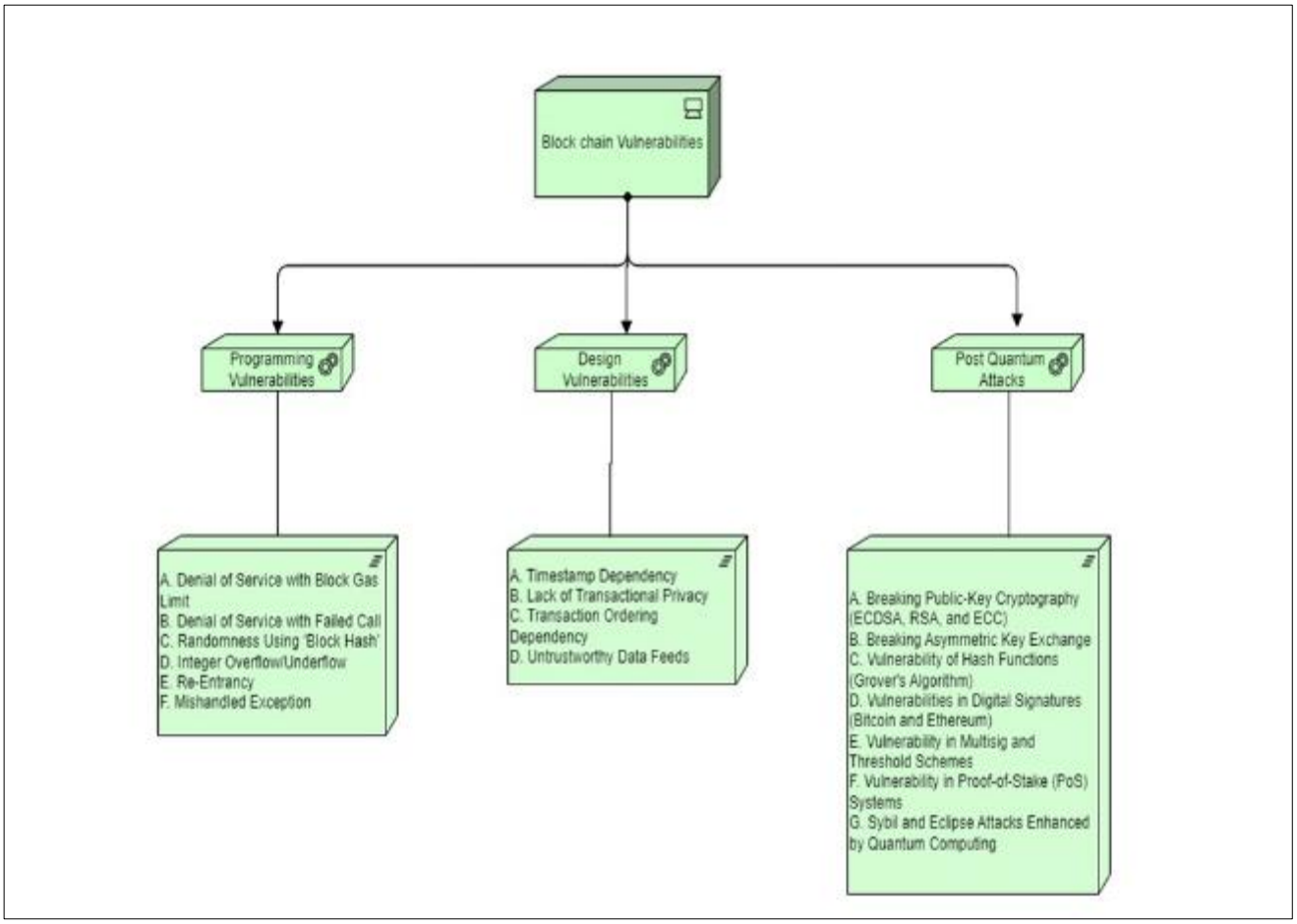

Figure 2 Blockchain Vulnerabilities Classification

3.1. Programming Language Vulnerabilities

3.1.1. Denial of Service with Block Gas Limit [32]

Each block in the Ethereum smart contract has an upper limiton the amount of gas (10,000,000 gas [33]) that can be spent for computation. Thus, if the gas spent on any transaction exceeds this block gas limit, then this leads to denial of service and the transaction will fail. This is the case with theunknown size of arrays which grow over time.

Preventive Method

Rather than modifying these arrays completely at once in a single block, several blocks should be taken. Because taking several blocks will break a single transaction into multiple transactions, hence it reduces the gas required for each transaction. This will reduce the riskof exceeding the block gas limit, and hence prevent denial ofservice.

3.1.2. Denial of Service with Failed Call [34]

External calls in a smart contract can fail accidentally (due to programming errors) or intentionally (by an attacker). In a situation when an attacker combines several calls in a singletransaction and executes in a loop, then this can prevent othersmart contracts nodes to interact with it.

Preventive Method

The contract logic to handle failed calls should be executed in such a way that several calls of Ether transfer must not be combined in a single transaction with the assumption that the external calls will always failto choose 'pull' over 'push' for external calls. It should implement contract logic to handle failed calls.

3.1.3. Randomness Using 'Block Hash' [9], [35]

Sometimes randomness is required in operations and block hash can be used for this purpose. However, it can be manipulated like timestamp because the same can be predictable by the miners. The use of 'block hash' as a source of randomness is shown in Figure 10 solidity code.

Preventive Method

The block hash can be read by any other transaction within the same mined block. If the attackeris also a miner, then it can be manipulated and worse can happen. Therefore, the external sources of randomness should be used, for example, Oracle, RNG [36], RANDAO, etc.

3.1.4. Integer Overflow/Underflow [37], [38]

This kind of situation will occur when the value in a calculation exceeds the lower or upper range of the variable size type, so cannot be expressed by that type. This can bethe case with smaller data types. If the balance will be at upper limit unit value (2^{256}), then it will reset againto zero. An unknown hacker drained off 2000 ether (cryptocurrency coin of the Ethereum blockchain) which wasworth $2.3 million using this vulnerability from ''Proof of Weak Hands Coin'' (PoWHC), which was a legit Ponzi scheme smart contract.

Preventive Method

To tackle integer overflow and under-flow issues, arithmetic operations should be implemented very carefully by comparing the operators and operandsbefore the operation. It is suggested to use assert(), require()functions, and 'SafeMath.sol' [38] library for arithmeticfunctions.

3.1.5. Re-Entrancy [40], [41], [42]

It is also known as a recursive call attack. In re-entrance vulnerability, a malicious contract calls back into the calling contract beforethe first invocation of the function is finished. Therefore, due to this recursive nature of the call, the malicious user can do multiple repetitive withdrawals without affecting hisbalance. An attacker stole 60 million US dollars by utilizing this vulnerability in the Decentralized Autonomous Organizations (DAO).

Preventive Method

For protecting against re-entrancy attacks, a reliable method can be used such as the ''Check-Effect-Interaction pattern''. This defines the order in whichwe should structure our functions to assure all internal statechanges before the next call. After resolving all state changes,the function should be allowed to interact with other con-tracts. To

prevent cross-function re-entrance attacks, mutexlocks are suggested to use. Any contract's state can be lockedusing a mutex lock and can only be modified by the owner ofthe lock. It prevents the recursive call of 'withdraw function'by an attacker. Open Zeppelin has its mutex implementationcalled as Re-entrance guard which acts as a re-entrancy lock.

3.1.6. Mishandled Exception [43]

When a contract is called by another contract and an exception or error is raisedin the Calle contract. But if the same might not be reportedto the Calle contract, then it might lead to threats. Therefore,it can attract the attackers to execute the malicious code in the contract.

Preventive Method

An exception in the callee may or maynot be propagated to the caller depending on how the functionwas called. Exception handling operators should be used toavoid such types of situations. The return value of functionsmust always be checked and an exception should be thrown.

3.1.7. Blockchain Design Vulnerabilities

Timestamp Dependency [9]

It is classified as asecurity vulnerability by Maher and Alharby [44]. It leads to a vulnerable situation when the triggered condition to execute the transaction is the block timestamp, because the dishonestminers can utilize the block timestamp value in an unethicalway.

Preventive Method

Luu *et al.* [8] suggested to use block number instead of block timestamp. Because the block number cannot be altered by the malicious miner. Therefore, it issuggested not to assign a block timestamp to a variable in thesmart contract code.

3.1.8. Lack of Transactional Privacy [44], [46]

Transactions balance details of the users are publicly available. But users want that their financial details and transactions should not be visible to others. It may limit the users ofsmart contracts since attackers can monitor the transaction- related details of users. Attackers can use this information forvarious kinds of unethical uses.

Preventive Method

Watanabe *et al.* [21] suggested theencryption of the smart contracts before deploying them on the blockchain. Kosba *et al.* [46] developed a tool to create aprivacy-preserving contract. They included the important features of privacy protection not only in Ethereum blockchain-based smart contract applications but also for all types of blockchains. As a preventive method to this issue, they implemented a decentralized smart contract framework, Hawk, which does not store financial transactions in the blockchainand spares the developers from implementing any crypto-graphic function.

3.1.9. Transaction Ordering Dependency [47]

This vulnerability is related to the execution order of two dependent transactions that are invoking the same smart contract. A malevolent user can utilize this vulnerability to attack if the transactions are not executed in the proper order. The order of transaction execution is decided by the miners, but if the adversary is the miner itself, then this will be a very disastrous situation.

Preventive Method

Natoli *et al.* [48] used Ethereum- based functions (e.g., SendIfReceived) to enforce the order of transactions. Luu *et al.* [10] recommended a guard conditionsuch that ''a contract code either returns the expected outputor fails''. To defend against Transaction Ordering Dependency attacks, Shuai Wang *et al.* [49] suggested a pre-commit scheme

3.1.10. Untrustworthy Data Feeds [50]

Some smart contracts need data feeds from outside the blockchain, but there is no guarantee that the external data source is trustworthy. So, in the case when an attacker is intentionally sending wrong information to fail the smart contract operation then it will be a hazardous situation.

Preventive Method

Zhang et al. [50] developed a tool named Town Crier (TC). TC works as a trustworthy more mediate in between the external source and a smart contract. In other words, it acts as an authenticator and provides data feed privacy using encryption. Town Crier tool comprises a Town Crier smart contract that dwells on the blockchain, and a Town Crier worker which lives exterior the blockchain. The data feed request can be sent to the Town Crier contract by the client contract, which finally will send to the Town Crier worker. The Town Crier worker at that point speaks to outside information sources using HTTPS to get the information. After getting the necessary information, the worker will advance those feed requests to the Town Crier contract, which finally will send to the client contract.

3.2. Quantum Computing Vulnerabilities

Quantum computing presents significant threats to blockchain systems, primarily due to its ability to break traditional cryptographic algorithms that are foundational to blockchain security. Here are the main vulnerabilities that quantum computers could cause in blockchains:

3.2.1. Breaking Public-Key Cryptography (ECDSA, RSA, and ECC)

Blockchains like Bitcoin and Ethereum rely on Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures, which are used to verify ownership of funds and authenticate transactions. Quantum computers, using Shor's algorithm, can efficiently break ECDSA, RSA, and ECC (Elliptic Curve Cryptography).

Impact

- **Transaction Forgery**: Quantum computers could derive private keys from public keys, allowing an attacker to forge digital signatures. They could then impersonate users, steal funds, or submit unauthorized transactions.
- **Loss of Privacy**: Attackers could also break the privacy of blockchain users by revealing their private keys, undermining anonymity.

Cause

Shor's algorithm can solve the discrete logarithm problem (which underpins ECC) in polynomial time. A sufficiently large quantum computer can therefore reverse-engineer private keys from public keys, which in classical computing would take an impractically long time.

3.2.2. Breaking Asymmetric Key Exchange

Asymmetric encryption methods used for secure communication between blockchain nodes (e.g., RSA, DH, and ECDH key exchanges) are also vulnerable to quantum attacks via Shor's algorithm.

Impact

- **Compromised Network Communication**: Quantum computers could decrypt peer-to-peer encrypted communication, exposing sensitive data (such as private transaction details, smart contract interactions, and validator communication) between blockchain nodes.
- **Attacking Consensus Mechanisms**: Certain consensus algorithms (e.g., in Proof-of-Stake or Proof-of-Authority systems) may rely on secure communication channels for leader election or validator verification. Breaking this could enable attackers to disrupt consensus, censor blocks, or manipulate governance systems.

Cause

Shor's algorithm can also factor large integers and solve the discrete logarithm problem used in Diffie-Hellman key exchange, exposing session keys that encrypt communications.

3.2.3. Vulnerability of Hash Functions (Grover's Algorithm)

Blockchains use hash functions (e.g., SHA-256 in Bitcoin) for mining, creating cryptographic links between blocks, and generating addresses. While hash functions are relatively secure, Grover's algorithm can reduce the security of hash functions by quadratically speeding up brute-force attacks.

Impact

- **Double-Spending Attacks**: With faster hash collision attacks, quantum computers could create collisions more efficiently, enabling double-spending or block forging, especially in systems with lower difficulty thresholds or small block intervals.
- **Undermining Proof-of-Work**: In proof-of-work (PoW) blockchains, quantum computers could gain a significant advantage in mining by solving hash puzzles faster than classical miners. This would allow quantum miners to dominate mining rewards or perform 51% attacks more easily.

Cause

Grover's algorithm reduces the time to find a pre-image (i.e., brute-forcing a hash) from $O(2^n)$ to $O(2^{(n/2)})$, effectively halving the security of hash-based systems.

3.2.4. Vulnerabilities in Digital Signatures (Bitcoin and Ethereum)

When users reuse addresses or expose their public keys (which happens after any transaction), quantum computers could use those public keys to derive the corresponding private keys, leading to significant security risks.

Impact

- **Retroactive Attacks**: An attacker could exploit historical transactions where public keys have already been exposed (e.g., Bitcoin UTXO model). Once a quantum computer is capable, attackers could retroactively derive private keys and steal funds from wallets or accounts.
- **Account Takeovers**: In Ethereum, public keys are often visible in smart contract interactions. This would make Ethereum accounts especially vulnerable to quantum attacks.

Cause

The mathematical problems that ensure the security of ECDSA and RSA (i.e., the discrete logarithm problem and integer factorization) can be easily solved by quantum computers using Shor's algorithm.

3.2.5. Vulnerability in Multisig and Threshold Schemes

Multisignature (multisig) wallets and threshold signature schemes are often used in decentralized finance (DeFi), custodial systems, or as governance mechanisms. These schemes rely on cryptographic security for group verification and asset control.

Impact

- **Compromise of Multisig Wallets**: A quantum computer could derive the private keys of individual signers in a multisig wallet, allowing an attacker to approve transactions and access funds without permission from other signers.
- **Governance Takeover**: In decentralized autonomous organizations (DAOs) or blockchains using threshold signature schemes for governance or voting, quantum computers could tamper with voting processes by forging signatures.

Cause

The same weaknesses in public-key cryptography apply to the individual private keys involved in multisig wallets or threshold schemes.

3.2.6. Vulnerability in Proof-of-Stake (PoS) Systems

Proof-of-Stake systems rely on validators staking funds and signing blocks using cryptographic keys. If quantum computers can break these cryptographic keys, the entire staking mechanism could be compromised.

Impact

- **Validator Takeover**: Attackers could hijack validator keys, allowing them to forge blocks or double-sign, potentially leading to slashing penalties or loss of funds.
- **Consensus Manipulation**: If a quantum attacker can control a sufficient number of validators by compromising their private keys, they could disrupt the consensus process, perform censorship, or execute **long-range attacks** (by rewriting long parts of the blockchain history).

Cause

The reliance on ECDSA signatures for validator identification and block signing is quantum-vulnerable. If an attacker can derive private keys, they could control validator roles in PoS blockchains.

3.2.7. Sybil and Eclipse Attacks Enhanced by Quantum Computing

Quantum-enhanced attacks on cryptographic keys could make it easier for an attacker to control multiple identities or nodes in the network, exacerbating Sybil or eclipse attacks.

Impact

- **Network Partitioning**: In an eclipse attack, an attacker can isolate a node and control all communication to and from it. With quantum computing, forging cryptographic keys could allow an attacker to take over nodes or masquerade as many legitimate nodes in a Sybil attack.
- **Decentralization Threats**: In a PoS system or DPoS system, a quantum-powered Sybil attack could allow an attacker to artificially inflate their stake or control a disproportionate number of delegate nodes.

Cause

By breaking cryptographic keys, attackers can create multiple forged identities or take over existing ones to launch these attacks.

4. Security Analysis Tools

4.1. Oyente

Oyente is one of the first tools created for analyzing Ethereum smart contracts, specifically written in Solidity. It performs symbolic execution of the bytecode and identifies common security issues. Oyente emulates the Ethereum Virtual Machine (EVM) and symbolically executes smart contracts to detect potential vulnerabilities. The symbolic execution checks all possible paths a program could take, helping to uncover problems like:

- Transaction order dependency
- Reentrancy attacks
- Time dependency
- Integer overflows and underflows
- Mishandled exceptions

4.2. Securify

Securify is another popular tool for smart contract analysis with a different approach. It focuses on automatically verifying whether smart contracts comply with a predefined set of security guidelines.

Securify relies on pattern matching to check contracts against security rules. It breaks the contract down into control flows and analyzes the code structure based on both compliance and violation patterns. Unlike Oyente, which uses symbolic execution, Securify uses static analysis methods. It checks for Access control violations, Incorrect use of block information (such as block. Timestamp), Integer overflow/underflow.

4.3. SmartCheck

This tool is a static analysis tool that uses ANTLR [52] (a parser generator) to translate Solidity source code into an XML parse tree [53] (an XML-based intermediate representation), and checks it against XPath [54] patterns [55].

4.4. Defect Checker

The defect Checker tool takes bytecodes as input, disassembles them into opcodes, splits the opcodes into several basic blocks and symbolically executes instructions in each block [56]. Then it generates the control flow graph (CFG) and records all stack events. Using CFG and stack events information, it detects three pre-defined features: money call, loop block and payable function. After feature detection, it applies rules to detect eight vulnerabilities: transaction state dependency, DoS under external influence, strict balance equality, re-entrancy, nested call, greedy contract, unchecked external calls, and block info dependency.

4.5. Contract Ward

The Contract Ward applies supervised learning to find vulnerabilities [56].It extracts 1619 dimensional bigram features from opcodes using an n-gram algorithm [57] and forms a feature space. Then it labels contracts in training set with six types of vulnerabilities using Oyente [58]. The label is stored in a six-dimension vector (e.g., [1 0 1 0 1 1]) where each bit stands for an existing vulnerability. Based on the feature space and labels of the training set, Contract Ward uses five classification algorithms to detect vulnerabilities.

4.6. NP Checker

This tool analyzes the non-determinism in the smart-contract execution context and then performs systematic modelling to expose various non-deterministic factors in the contract execution context [59]. Non-deterministic factors are factors that could impact final results to the end-user and make them unforeseeable. Possible factors discussed in NPChecker are block and transaction state, transaction execution scheduling, and external callee. The NPChecker disassembles the EVM bytecode and translates them into LLVM intermediate representation (IR) code [64], recovers the control flow structures and enhances the LLVM IR with function information, identifies state and global variables, and performs information-flow tracking to analyze their influences on the fund's transfer.

4.7. MadMax

The MadMax tool uses the Vandal decompiler [64] to produce CFG, a three-address code for all operations in the smart contract, and function boundaries. Then it analyzes the three-address-code representation, recognizes concepts such as loops and induction variables, analyzes memory and dynamic data structures, and infers the concept of gas-focused vulnerabilities. This tool only detects gas-focused vulnerabilities, such as unbounded mass operations (infinite or nearly infinite loops), external calls that throw out-of-gas exceptions or arithmetic integer overflows, because those vulnerabilities will cause unexpected gas consumption.

4.8. Osiris

The Osiris combines symbolic analysis and taint analysis technology to find integer bugs in smart contracts [56]. It has three components: symbolic analysis, taint analysis and integer error detection. The symbolic analysis component creates CFG and symbolical executions of every path in the CFG. Then, the taint analysis part checks for taints across the stack, memory and storage, and integer error detection looking for possible integer bugs within the executed instruction.

4.9. Contract Fuzzer

The contractFuzzer tool combines static analysis of ABI and bytecodes and fuzzing technology to explore vulnerabilities of smart contracts [64]. It creates an Ethereum test net using offline EVM to monitor the execution of the smart contracts and extract information from the execution process. By analyzing the ABIs and bytecodes, contractFuzzer calculates the function selector (first four bytes of the hash of the function's signature) and maps each ABI function to a set of function selectors used. Then an input generation algorithm is created to fuzz each function based on the information of the previous step. It collects three types of test oracles during the execution of smart contracts: attributes of a contract call or delegate call, run-time information about opcodes invoked, and the state of the contract.

4.10. Sereum

It is a modified Ethereum client based on Geth that focuses on re-entrancy vulnerabilities cross-function re-entrancy, delegated re-entrancy, and create-based re-entrancy. The cross-function re-entrancy is to re-enter another function in the same contract. The delegated re-entrancy is to re-enter a smart contract via delegate call to an unsafe library that may use address.call().value() to call back to the original contract. The create-based re-entrancy utilizes the fact that a newly created contract will have its constructor function executed immediately when the contract is deployed. The constructor is deemed safe and trusted, but it could contain external calls to malicious contracts. If the victim contract creates another contract in function A and calls the attacker's contract in the newly created contract's constructor, the attacker's contract could re-enter the victim contract by calling function A. Sereum adds two components to the Geth client: taint engine (taints and tracks state variables along with the executions of various functions that detects conditional JUMP instructions influenced by a storage variable), and attack detector (creates locks that prohibit further updates for state variables that influence control flows).

4.11. sFuzz

The sFuzz tool is an adaptive fuzzer that combines the strategy in the AFL fuzzer [64] and other adaptive strategies built on Aleth with implementations of three more components: runner, libfuzzer, and liboracles. The runner sets up the test net environment and options of the other two components. Then the contracts are executed on the test net, where transactions are generated based on the analysis of the contract's ABI.

- The libfuzzer selectively generates test cases by implementing a feedback-guided adaptive fuzzing strategy.
- The liboracles monitors the execution of a test case and the corresponding stack events to check for vulnerabilities.

Table 1 Compatibility Matrix for Vulnerabilities and Security Analysis Tools

Security Tool	Re-Entrancy	Arithmetic Issues	Delegate call to Insecure Contracts	Self destruct	Tx.origin	Mishandled Exceptions	External Contract Referencing	Short Address	Freezing Ether	Transaction Order Dependence	Generating Randomness	Timestamp Dependence
Oyente	C					C				C		C
Securify	C		C			C		C	C	C		
SmartCheck	C	C			C				C			C
DefectChecker	C					C			C	C		C
ContractWard	C	C								C		C
NPChecker	C									C		C
MadMax		C										
Osiris		C										
ContractFuzzer	C		C	C		C			C			C
Sereum	C											
sFuzz	C	C	C	C		C			C			C

5. Public Key Post Quantum Cryptosystems

Public Key Post Quantum Cryptosystems can be broadly categorized into 5 major types namely

- Code-based cryptosystems,
- Multi-Variate based cryptosystems,
- Lattice based cryptosystems,
- Super Singular Elliptic based Isogeny and
- Hybrid cryptosystems.

As a summary, the five different types of post-quantum cryptosystems are depicted in figure together with examples of encryption and digital signature scheme implementations.

There are four main types of post-quantum cryptosystems and a fifth kind that actually mixes both pre-quantum and post-quantum cryptosystems. The following subsections analyze the potential application of such schemes for the implementation of encryption/decryption mechanisms and for signing blockchain transactions.

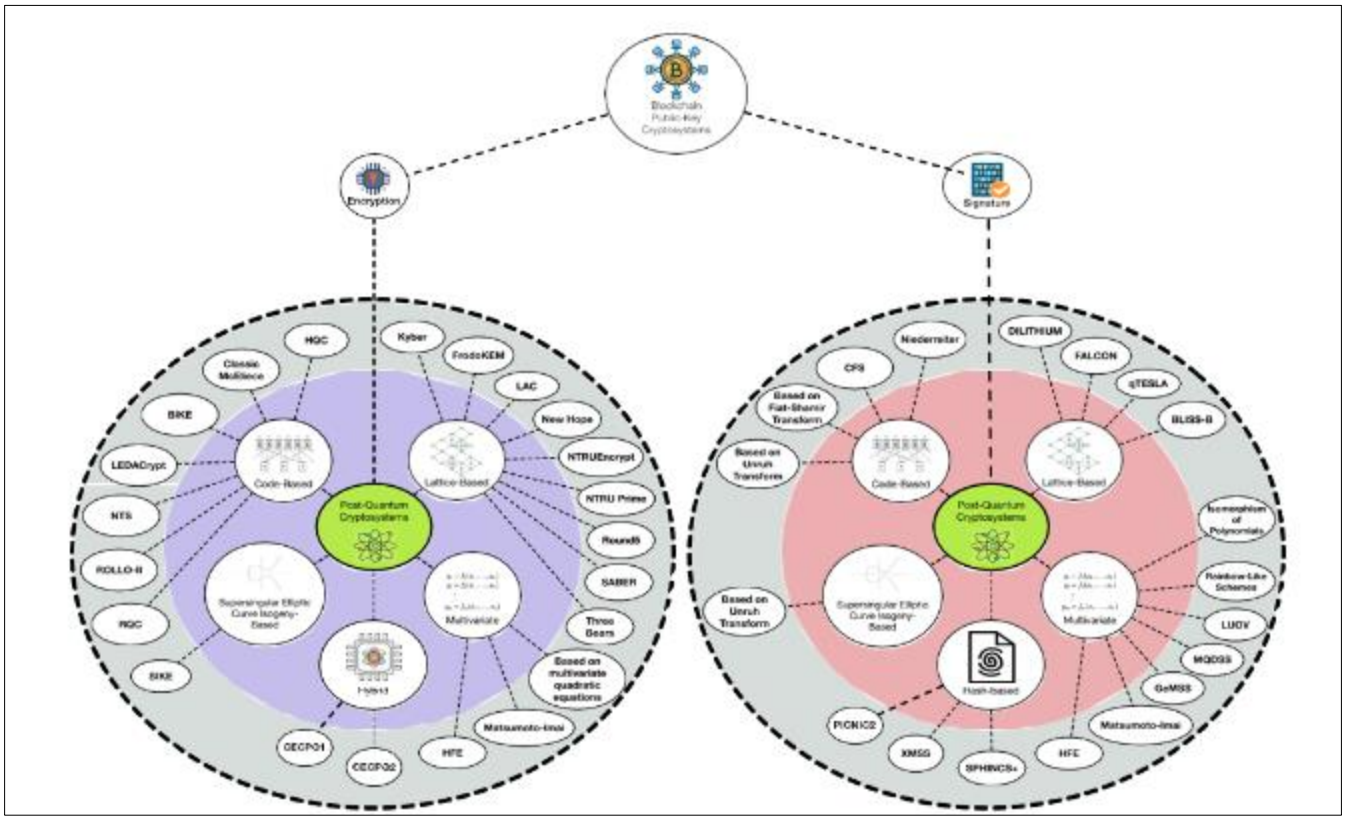

Figure 3 Blockchain Post Quantum Public – Key Cryptosystems

5.1.1. Code-Based Cryptosystems

They are essentially based on the theory that supports error- correction codes. For instance, McEliece's cryptosystem is an example of code-based cryptosystem [63] those dates back from the 70s and whose security is based on the syndrome decoding problem. McEliece's scheme provides fast encryption and relatively fast decryption, which is an advantage for performing rapid blockchain transactions. However, McEliece's cryptosystem requires to store and perform operations with large matrices that act as public and private keys. Such matrices usually occupy between 100 kilobytes and several megabytes, which may be a restriction when resource- constrained devices are involved. To tackle this issue, future researchers will have to study matrix compression techniques, as well as the use of different codes (e.g., Low- Density Parity-Check (LDPC) codes, Quasi-Cyclic Low- Rank Parity-Check (QC-LRPC) codes) and specific coding techniques [63].

5.1.2. Multi-Variate Based Cryptosystems

Multivariate-based schemes rely on the complexity of solving systems of multivariate equations, which have been demonstrated to be NP-hard or NP-complete. Despite their resistance to quantum attacks, it is necessary further research for improving their decryption speed (due to the involved ''guess work'') and to reduce their large key size and cipher- text overhead [64].

Currently, some of the most promising multivariate-based schemes are the ones based on the use of square matrices with random quadratic polynomials, the cryptosystems derived from Matsumoto-Imai's algorithm and the schemes that rely on Hidden Field Equations (HFE) [65] – [67].

5.1.3. Lattice - Based Cryptosystems

This kind of cryptographic schemes are based on lattices, which are sets of points in n-dimensional spaces with a periodic structure. Lattice-based security schemes rely on the presumed hardness of lattice problems like the Shortest Vector Problem (SVP), which is an NP-hard problem whose objective is to find the shortest non-zero vector within a lattice. There are other similar lattice-related problems like the Closest Vector Problem (CVP) or the Shortest Independent Vectors Problem (SIVP) [68], which nowadays cannot be solved efficiently through quantum computers.

Lattice-based schemes provide implementations that allow for speeding up blockchain user transactions since they are often computationally simple, so they can be executed fast and in an efficient way. However, like it occurs with other post-quantum schemes, lattice-based implementations need to store and make use of large keys, and involve large ciphertext overheads. For example, lattice-based schemes like NTRU or New Hope often require to manage keys in the order of a few thousand bits.

As of writing, the most promising lattice-based cryptosystems are based on polynomial algebra [70], [71] and on the Learning With Errors (LWE) problem and its variants (e.g., LP-LWE (Lindner-Peikert LWE) or Ring-LWE [72]).

5.1.4. Super-Singular Elliptic Curve Isogeny Cryptosystems

These schemes are based on the isogeny protocol for ordinary elliptic curves presented in [72], but enhanced to withstand the quantum attack detailed in [73]. There are different promising post-quantum cryptosystems of this type, whose key size is usually in the order of a few thousand bits .

Only one isogeny-based public-key encryption scheme passed to the second round of the NIST call: SIKE. SIKE is based on pseudo-random walks in supersingular isogeny graphs. A good reference of SIKE key sizes is SIKEp434, which, for a 128-bit level of classical security, makes use of a 2640-bit public-key and a 2992-bit private key.

5.1.5. Hybrid Cryptosystems

Hybrid schemes seem to be next step towards post-quantum security, since they merge pre-quantum and post-quantum cryptosystems with the objective of protecting the exchanged data both from quantum attacks and from attacks against the used post-quantum schemes, whose security is currently being evaluated by industry and academia.

This kind of cryptosystems have been tested by Google [76], which merged New Hope with an ECC-based Diffie-Hellman key agreement scheme named X25519. A second version of the hybrid scheme (CECPQ2) is currently being tested: it merges X25519 with instantiations of NTRU (HRSS (Hülsing, Rijneveld, Schanck, Schwabe) and SXY (Saito, Xagawa, Yamakawa)).

Although these schemes look promising, it must be noted that they involve implementing two complex cryptosystems, which require significant computational resources and more energy consumption. Therefore, future developers of hybrid post-quantum cryptosystems for blockchains will have to look for a trade-off between security, computational complexity and resource consumption. In addition, developers will have to address the large payload problem that arises with this kind of cryptosystems when providing Transport Layer Security (TLS) communications (such a problem is due to the required public-key and ciphertext sizes).

6. Post-Quantum Signing Algorithms

6.1.1. Code-Based Signing Algorithms

Different post-quantum code-based signing algorithms have been proposed in the past. Some of the most relevant subtypes of this kind of cryptosystems are based on the schemes from Niederreiter [77] and CFS (Courtois, Finiasz, Sendrier) , which are really similar to McEliece's cryptosystem. The signatures of such schemes are short in length and can be verified really fast, but, as it occurs with traditional McEliece's cryptosystems, the use of large key sizes requires significant computational resources and, as a consequence, signature generation may become inefficient.

Other code-based signing algorithms have been proposed in the literature, such as identification protocols related to the application of Fiat-Shamir transformation [78], which in some cases outperform cryptosystems like CFS . Nonetheless, it must be noted that, Fiat-Shamir signatures are not known to be completely secure against quantum, so alternatives like the Unruh transformation should be considered.

6.1.2. Multi-Variate Code-Based Signing Algorithms

In this kind of signature schemes the public key is generated through a trapdoor function that acts as private key. This fact usually derives into large public keys, but very small signatures.

Some of the most popular multivariate-based schemes rely on Matsumoto-Imai's algorithm, on Isomorphism of Polynomials (IP) [79] or on variants of HFE, which are able to generate signatures with a size comparable to the currently used RSA or ECC-based signatures [80]. Other relevant multivariate-based digital signature schemes have been proposed, like the ones based on pseudo-random multi- variate quadratic equations or on Rainbow-like signing schemes (e.g., TTS, TRMS or Rainbow). Nonetheless, such cryptosystems need to be further improved in terms of key size, since they usually require several tens of thousands of bytes per key.

6.1.3. Lattice-Based Signing Algorithms

Among the different lattice-based signature schemes described in the literature, the ones based on Short Integer Solution (SIS) seem to be promising due to their reduced key size. According to some performance analyses, BLISS-B (Bimodal Lattice Signatures B), which relies on the hardness of the SIS problem, provides one of the best performances for lattice-based signing cryptosystems, being on a par with RSA and ECDSA . However, note that the original BLISS was attacked in 2016 under specific conditions through a side-channel attack, while its variant BLISS-B is also susceptible to cache attacks that are able to recover the secret signing key after 6,000 signature generations.

Besides BLISS, there are in the literature other lattice- based signature schemes that rely on the SIS problem but that were devised specifically for blockchains [81]. Researchers have also developed lattice-based blind sig- nature schemes, which were introduced by David Chaum in the early 80s for creating an untraceable payment system [82]. For instance, a lattice-based blind signature scheme is detailed in, which was specifically conceived for providing user anonymity and untraceability in distributed blockchain-based applications for IoT.

Finally, it is worth mentioning the lattice-based signature schemes presented in. Specifically, in the authors propose a cryptosystem whose public and private keys are generated through Bonsai Trees. Regarding the work in, it presents a lattice-based signature scheme optimized for embedded systems, which, for a 100-bit security level, makes use of a public key of 12,000 bits and a private key of 2,000 bits, and generates signatures of 9,000 bits. This latter scheme, due to its simplicity and efficiency, was selected as signature algorithm for blockchain-related developments like QChain, a post-quantum decentralized system for managing public-key encryption.

6.1.4. Super-Singular Elliptic Curve Isogeny Signature Schemes

It is possible to use supersingular elliptic curve isogenies for creating post-quantum digital signature schemes, but there are not in the literature many of such schemes and they still suffer from poor performance. For instance, in [87] the authors present different signature schemes based on isogeny problems and on the Unruh transform, which makes use of small key sizes and relatively efficient signing and verification algorithms. Another signature scheme based on the Unruh transform is presented in, which, for a 128-bit quantum security level, makes use of a 336-byte public key and a 48-byte private key, but it generates 122,880-byte signatures (even when using compression techniques). Therefore, it is necessary to address key size issues when implementing isogeny-based cryptosystems and Supersingular Isogeny Diffie-Hellman (SIDH), especially in the case of resource-constrained devices, which need to use key compression techniques that often involve computationally intensive steps.

6.1.5. Hash–Based Signature Schemes

The security of these schemes depends on the security of the underlying hash function instead of on the hardness of a mathematical problem. This kind of schemes date back from the late 70s, when Lamport proposed a signature scheme based on a one-way function. Currently, variants of eXtended Merkle Signature Scheme (XMSS) like XMSS-T and SPHINCS [88] are considered promising hash-based signature schemes for the post-quantum era that derive from the Merkle tree scheme described in.

However, some researchers consider XMSS and SPHINCS to be impractical for blockchain applications due to their performance, so alternatives have been suggested. For example, XMSS has been adapted to blockchain by making use of

a single authentication path instead of a tree, while using one-time and limited keys in order to pre- serve anonymity and minimize user tracking. Other authors proposed substituting XMSS with XNYSS (eXtended Naor-Yung Signature Scheme), a signature scheme that combines a hash-based one-time signature scheme with Naor-Yung chains, which allow for creating chains of related signatures [89].

7. Conclusion and Future Enhancements

Quantum computers pose a serious threat to cryptographic foundations of blockchains, especially relying on classical public-key algorithm and hash-based functions. Future-proofing blockchain cryptography by integrating post-quantum cryptography is essential.

- Coded-based cryptosystems make use of large keys whose management and operation require a relevant number of computational resources. More research is necessary on key compression techniques and on the use of certain types of codes and coding techniques.
- Lattice-based cryptosystems also need to be enhanced in terms of key size, but it can be stated that they are currently some of the most promising candidates for implementing schemes for post- quantum blockchains.
- Multivariate-based public-key cryptosystems still need to be improved to increase decryption speed and to decrease key size. However, it should be noted that some multivariate-based signature algorithms optimized for the AVX2 instruction set (i.e., LUOV, MQDSS and Rainbow) are clearly faster than most of the compared digital signature cryptosystems.
- Hybrid schemes like the ones tested by Google (CECPQ1 and CECPQ2) seem to be the next step prior to the actual implementation of pure post- quantum blockchains, but they require to make use of hardware able to handle at the same time two advanced security mechanisms and large payloads.
- Super-singular elliptic-curve isogeny cryptosystems based on the Unruh transform seem promising, but still need to be optimized to decrease their signature size.
- Hash-based digital signature cryptosystems have in general poor performance, but some researchers have suggested new faster algorithms that seem to be practical for blockchain [89]

Compliance with ethical standards

Disclosure of conflict of interest

No competing interests related to this publication.

References

[1] A. Averin and O. Averina, ‘‘Review of blockchain technology vul-nerabilities and blockchain-system attacks,’’ in *Proc. Int. Multi-Conf. Ind. Eng. Mod. Technol.*, Oct. 2019, pp. 1–6, doi: 10.1109/FarEast- Con.2019.8934243.

[2] C. S. Wright, ‘‘Bitcoin: A peer-to-peer electronic cash system,’’ *SSRN Electron. J.*, pp. 1–9, Jan. 2019, doi: 10.2139/ssrn.3440802.

[3] C. K. Frantz and M. Nowostawski, ‘‘From institutions to code: Towards automated generation of smart contracts,’’ in *Proc. IEEE 1st Int. Work- shops Found. Appl. Self Syst.*, Sep. 2016, pp. 210–215.

[4] M. In and F. Of, ‘‘Dubai aims to be a city built on blockchain where financial regulation goes in a republican era,’’ *Wall Str. J.*, vol. 4, pp. 3–6,Apr. 2017. [Online]. Available: https://www.wsj.com/articles/dubai- aims-to-be-a-city-built-on-blockchain-1493086080

[5] S. Kim and G. C. Deka, *Advanced Applications of Blockchain Technology*,vol. 60. Singapore: Springer, 2020.

[6] M. Singh and S. Kim, ‘‘Blockchain technology for decentralized autonomous organizations,’’ in *Proc. Adv. Comput.*, vol. 115, Oct. 2019,pp. 115–140.

[7] C. E. Brown, O. Kuncar, and J. Urban, ‘‘Formal verification of smart contracts,’’ in *Proc. ACM Workshop Program. Lang. Anal. Secur.*, 2017,pp. 91–96. [Online]. Available: http://proofmarket.org

[8] M. Bartoletti and L. Pompianu, ‘‘An empirical analysis of smart contracts: Platforms, applications, and design patterns,’’ in *Financial Cryptographyand Data Security* (Lecture Notes in Computer Science), vol. 10323. Cham, Switzerland: Springer, 2017, pp. 494–509, doi: 10.1007/978-3- 319-70278-0_31.

[9] M. Wöhrer and U. Zdun, ‘‘Design patterns for smart contractsin the ethereum ecosystem,’’ in *Proc. IEEE Int. Conf. Internet Things*, Aug. 2018, pp. 1513–1520, doi: 10.1109/Cybermat- ics_2018.2018.00255.

[10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, ‘‘Makingsmart contracts smarter,’’ in *Proc. ACM SIGSAC Conf. Comput. Commun.Secur.*, Oct. 2016, pp. 254–269, doi: 10.1145/2976749.2978309.

[11] A. Mense and M. Flatscher, ‘‘Security vulnerabilities in ethereum smart contracts,’’ in *Proc. 20th Int. Conf. Inf. Integr. Appl. Services*, Nov. 2018,pp. 375–380, doi: 10.1145/3282373.3282419.

[12] D. Bayer, S. Haber, and W. S. Stornetta, ‘‘Improving the efficiencyand reliability of digital time-stamping,’’ in *Proc. Sequence*, 1993, pp. 329–334, doi: 10.1007/978-1-4613-9323-8_24.

[13] R. Merkel, *Secrecy, Authentication and Public Key Systems*. Stanford, CA, USA: Stanford Univ., 1979, p. 193.

[14] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang, ‘‘An overview of smart contract: Architecture, applications, and future trends,’’ in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 108–113, doi: 10.1109/IVS.2018.8500488.

[15] (2018). *Eos*. [Online]. Available: https://eos.io/

[16] (2017). *Tron*. [Online]. Available: https://tron.network/

[17] B. Canou, G. Henry, and P. Chambart, ‘‘Tezos: The ocaml crypto-ledger,’’in *Proc. OCaml Users Developers Workshop*, 2017, pp. 1–2.

[18] B. A. Kitchenham, D. Budgen, and O. P. Brereton, ‘‘Using mapping studies as the basis for further research—A participant-observer case study,’’ *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 638–651, Jun. 2011, doi: 10.1016/j.infsof.2010.12.011.

[19] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, ‘‘Systematic mappingstudies in software engineering,’’ in *Proc. Electron. Workshops Comput.*,Jun. 2008, pp. 1–10, doi: 10.14236/ewic/ease2008.8.

[20] Dika, A. Ethereum Smart Contracts: Security Vulnerabilities and Security Tools. Master’s Thesis, Norwegian University of Science and Technology, Trondheim, Norway, December 2017.

[21] Luu, L.; Chu, D.H.; Olickel, H.; Saxena, P.; Hobor, A. Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS’16), Vienna, Austria, 24–28 October 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 254–269

[22] Zheng, Z.; Xie, S.; Dai, H.N.; Chen, W.; Chen, X.; Weng, J.; Imran, M. An overview on smart contracts: Challenges, advances and platforms. Future Gener. Comput. Syst. 2020, 105, 475–491

[23] Atzei, N.; Bartoletti, M.; Cimoli, T. A survey of attacks on ethereum smart contracts (sok). In Proceedings of the International Conference on Principles of Security and Trust, Uppsala, Sweden, 24–25 April 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 164–186.

[24] Bartoletti, M.; Pompianu, L. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In Proceedings of the International Conference on Financial Cryptography and Data Security, Sliema, Malta, 3–7 April 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 494–509.

[25] He, D.; Deng, Z.; Zhang, Y.; Chan, S.; Cheng, Y.; Guizani, N. Smart Contract Vulnerability Analysis and Security Audit. IEEE Netw. 2020, 34, 276–282. [CrossRef]

[26] Meiklejohn, S.; Pomarole, M.; Jordan, G.; Levchenko, K.; McCoy, D.; Voelker, G.M.; Savage, S. A fistful of bitcoins: Characterizing payments among men with no names. In Proceedings of the 2013 Conference on Internet Measurement Conference, Barcelona, Spain, 23–25 October 2013; pp. 127–140.

[27] Ron, D.; Shamir, A. Quantitative analysis of the full bitcoin transaction graph. In Proceedings of the International Conference on Financial Cryptography and Data Security, Okinawa, Japan, 1–5 April 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 6–24.

[28] Zhou, Y.; Kumar, D.; Bakshi, S.; Mason, J.; Miller, A.; Bailey, M. Erays: Reverse Engineering Ethereum’s Opaque Smart Contracts. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1371–1385.

[29] Community, E. What Is Oracles? 2021. Available online: https://ethereum.org/en/developers/docs/oracles/#top (accessed on 19 January 2022).

[30] Community, E. Oracle Problems. 2021. Available online: https://docs.ethhub.io/built-on-ethereum/oracles/what-are-oracles/ (accessed on 19 January 2022).

[31] SmartContract Chainlink Ltd. 2021. Available online: https://chain.link/

[32] J. Krupp and C. Rossow, ''TEETHER: Gnawing at ethereum to automatically exploit smart contracts,'' in Proc. 27th USENIX Secur. Symp., 2018, pp. 1317–1333.

[33] S. O'Neal. (2020). Ethereum Miners Vote to Increase Gas Limit, Causing Community Debate. Accessed: Jul. 30, 2020.[Online]. Available: https://cointelegraph.com/news/ethereum-miners-vote-to- increase-gas-limit-causing-community-debate

[34] W. Chen, Z. Zheng, E. C. H. Ngai, P. Zheng, and Y. Zhou, ''Exploiting blockchain data to detect smart Ponzi schemes on ethereum,'' IEEE Access, vol. 7, pp. 37575–37586, 2019, doi: 10.1109/ACCESS.2019.2905769.

[35] I.-C. Lin and T.-C. Liao, ''A survey of blockchain security issues and challenges,'' Int. J. Netw. Secur., vol. 19, no. 5, pp. 653–659, Sep. 2017, doi: 10.6633/IJNS.201709.19(5).01.

[36] C. F. Torres, J. Schátte, and R. State, ''Osiris: Hunting for integer bugs in ethereum smart contracts,'' in Proc. 34th Annu. Comput. Secur. Appl. Conf., Dec. 2018, pp. 664–676, doi: 10.1145/3274694.3274737.

[37] E. Mik, ''Smart contracts: Terminology, technical limitations and real world complexity,'' Law, Innov. Technol., vol. 9, no. 2, pp. 269–300, Jul. 2017, doi: 10.1080/17579961.2017.1378468.

[38] Mythril. Accessed: Aug. 27, 2021. [Online]. Available: https://github. com/ConsenSys/mithril

[39] A. Singh, R. M. Parizi, Q. Zhang, K. K. R. Choo, and A. Dehghan- tanha, ''Blockchain smart contracts formalization: Approaches and chal- lenges to address vulnerabilities,'' Comput. Secur., vol. 88, Oct. 2020, Art. no. 101654, doi: 10.1016/j.cose.2019.101654.

[40] T. Min, H. Wang, Y. Guo, and W. Cai, ''Blockchain games: A sur- vey,'' in Proc. IEEE Conf. Games (CoG), Aug. 2019, pp. 1–8, doi: 10.1109/CIG.2019.8848111.

[41] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, ''ReGuard: Find- ing reentrancy bugs in smart contracts,'' in Proc. 40th Int. Conf. Softw. Eng., Companion, May 2018, pp. 65–68, doi: 10.1145/3183440.3183495.

[42] J.-W. Liao, T.-T. Tsai, C.-K. He, and C.-W. Tien, ''SoliAudit: Smart contract vulnerability assessment based on machine learning and fuzz testing,'' in Proc. 6th Int. Conf. Internet Things, Syst., Manage. Secur. (IOTSMS), Oct. 2019, pp. 458–465, doi: 10.1109/IOTSMS48152.2019.8939256.

[43] B. Ghaleb, A. Al-Dubai, E. Ekonomou, M. Qasem, I. Romdhani, and

[44] L. Mackenzie, ''Addressing the DAO insider attack in RPL's Internet of Things networks,'' IEEE Commun. Lett., vol. 23, no. 1, pp. 68–71, 2019, doi: 10.1109/LCOMM.2018.2878151.

[45] M. Alharby and A. V. Moorsel, ''Blockchain based smart contracts: A sys- tematic mapping study,'' in Proc. Comput. Sci. Inf. Technol., Aug. 2017, pp. 125–140, doi: 10.5121/csit.2017.71011.

[46] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, ''Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,'' in Proc. IEEE Symp. Secur. Privacy (SP), May 2016, pp. 839–858, doi: 10.1109/SP.2016.55.

[47] M. di Angelo and G. Salzer, ''A survey of tools for analyzing ethereum smart contracts,'' in Proc. IEEE Int. Conf. Decentralized Appl. Infrastruct. (DAPPCON), Apr. 2019, pp. 69–78, doi: 10.1109/DAPP- CON.2019.00018.

[48] S. Wang, C. Zhang, and Z. Su, ''Detecting nondeterministic payment bugs in ethereum smart contracts,'' in Proc. ACM Program. Lang., vol. 3, 2019, pp. 1–29, doi: 10.1145/3360615.

[49] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, ''Zerocash:

[50] Decentralized anonymous payments from bit- coin,'' in Proc. IEEE Symp. Secur. Privacy, May 2014, pp. 459–474, doi: 10.1109/SP.2014.36.

[51] C. Natoli and V. Gramoli, ''The blockchain anomaly,'' in Proc. IEEE 15th Int. Symp. Netw. Comput. Appl. (NCA), Oct. 2016, pp. 310–317, doi: 10.1109/NCA.2016.7778635.

[52] Parr, T. 2021. Available online: https://www.antlr.org/ (accessed on 19 January 2022).

[53] Aho, A.V.; Sethi, R.; Ullman, J.D. Compilers, principles, techniques. Addison Wesley 1986, 7, 9.

[54] W3C. 2011. Available online: https://www.w3.org/TR/xpath20/ (accessed on 19 January 2022).

[55] Tikhomirov, S.; Voskresenskaya, E.; Ivanitskiy, I.; Takhaviev, R.; Marchenko, E.; Alexandrov, Y. Smartcheck: Static analysis of ethereum smart contracts. In Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, Gothenburg, Sweden, 27 May–3 June 2018; pp. 9–16.

[56] Chen, J.; Xia, X.; Lo, D.; Grundy, J.; Luo, X.; Chen, T. DEFECTCHECKER: Automated Smart Contract Defect Detection by Analyzing EVM Bytecode. IEEE Trans. Softw. Eng. 2021, 1, 1. [CrossRef]

[57] Wang, W.; Song, J.; Xu, G.; Li, Y.; Wang, H.; Su, C. Contractward: Automated vulnerability detection models for ethereum smart contracts. IEEE Trans. Netw. Sci. Eng. 2020, 8, 1133–1144. [CrossRef]

[58] Wang, A.; Wang, H.; Jiang, B.; Chan, W. Artemis: An improved smart contract verification tool for vulnerability detection. In Proceedings of the 2020 7th International Conference on Dependable Systems and Their Applications (DSA), Xi'an, China, 28–29 November 2020; pp. 173–181.

[59] Cavnar, W.B.; Trenkle, J.M. N-gram-based text categorization. In Proceedings of the SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV, USA, 11–13 April 1994.

[60] Luu, L.; Chu, D.H.; Olickel, H.; Saxena, P.; Hobor, A. Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16), Vienna, Austria, 24–28 October 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 254–269. [CrossRef]

[61] T. M. Fernández-Caramès and P. Fraga-Lamas, "Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks," in IEEE Access, vol. 8, pp. 21091-21116, 2020, doi: 10.1109/ACCESS.2020.2968985.

[62] The State of Ethereum Smart Contracts Security: Vulnerabilities, Countermeasures, and Tool Support by Haozhe Zhou,Amin Milani Fard *ORCID and Adetokunbo Makanju J. Cybersecur. Priv. 2022, 2(2), 358-378; https://doi.org/10.3390/jcp2020019

[63] W. Lee, J.-S. No, and Y.-S. Kim, ''Punctured Reed–Muller code-based McEliece cryptosystems,'' *IET Commun.*, vol. 11, no. 10, pp. 1543–1548, Jul. 2017.

[64] A. Petzoldt, S. Bulygin, and J. Buchmann, ''Selecting parameters for the rainbow signature scheme,'' in Proc. PQCrypto, Darmstadt, Germany, May 2010, pp. 218–240.

[65] J. Ding, A. Petzoldt, and L.-C. Wang, ''The cubic simple matrix encryption scheme,'' in Proc. PQCrypto, Waterloo, ON, Canada, Oct. 2014,pp. 76–87.

[66] J. Ding, ''A new variant of the Matsumoto-Imai cryptosystem through perturbation,'' in Proc. Int. Workshop Public Key Cryptogr., Singapore, Mar. 2004, pp. 305–318.

[67] J. Ding and D. Schmidt, ''Cryptanalysis of HFEv and internal perturbation of HFE,'' in Proc. Int. Workshop Public Key Cryptogr., Les Diablerets, Switzerland, Jan. 2005, pp. 288–301.

[68] J. Blömer and S. Naewe, ''Sampling methods for shortest vectors, closest vectors and successive minima,'' in Proc. Int. Colloq. Automata, Lang., Program., Wrocław, Poland, Jul. 2007, pp. 65–77.

[69] D. Stehlé and R. Steinfeld, ''Making NTRU as secure as worst-case problems over ideal lattices,'' in Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn., Tallinn, Estonia, May 2011, pp. 27–47.

[70] G. S. Aujla, R. Chaudhary, K. Kaur, S. Garg, N. Kumar, and R. Ranjan, ''SAFE: SDN-assisted framework for edge–cloud interplay in secure healthcare ecosystem,'' IEEE Trans. Ind. Inf., vol. 15, no. 1, pp. 469–480, Jan. 2019.

[71] R. Lindner and C. Peikert, ''Better key sizes (and attacks) for LWE-based encryption,'' in Proc. Cryptographers' Track RSA Conf., San Francisco, CA, USA, Feb. 2011, pp. 319–339.

[72] V. Lyubashevsky, C. Peikert, and O. Regev, ''A toolkit for ring-LWE cryptography,'' in Proc. EUROCRYPT, Athens, Greece, May 2013, pp. 35–54.

[73] A. Rostovtsev and A. Stolbunov, ''Public-key cryptosystem based on isogenies,'' Cryptol. ePrint Arch., Tech. Rep. 2006/145, 2006.

[74] A. Childs, D. Jao, and V. Soukharev, ''Constructing elliptic curve isogenies in quantum subexponential time,'' J. Math. Cryptol., vol. 8, no. 1,pp. 1–29, Jan. 2014.

[75] J.-F. Biasse, D. Jao, and A. Sankar, A Quantum Algorithm for Com- putting Isogenies Between Supersingular Elliptic Curves (Lecture Notes in Computer Science), vol. 8885. Cham, Switzerland: Springer, 2014, pp. 428–442.

[76] Google Blog Google's Experiments With a Hybrid Cryptosystem. Accessed: Nov. 2, 2019. [Online]. Available: https://security.googleblog. com/2016/07/experimenting-with-post-quantum.html

[77] H. Niederreiter, ''Knapsack-type cryptosystems and algebraic coding theory,'' Problems Control Inf. Theory, vol. 15, no. 2, pp. 159–166, 1986.

[78] N. T. Courtois, M. Finiasz, and N. Sendrier, ''How to achieve a McEliece- based digital signature scheme,'' in Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur., Gold Coast, QLD, Australia, Dec.,2001, pp. 157–174.

[79] J. Patarin, ''Hidden fields equations (HFE) and isomorphisms of poly- nomials (IP): Two new families of asymmetric algorithms,'' in Proc. EUROCRYPT, Saragossa, Spain, May 1996, pp. 33–48.

[80] A. Petzoldt, M.-S. Chen, B.-Y. Yang, C. Tao, and J. Ding, ''Design princi- ples for HFEv-based multivariate signature schemes,'' in Proc. Int. Conf. Adv. Cryptol., Auckland, New Zealand, Nov./Dec. 2015, pp. 311–334.

[81] N. T. Curtois, ''On multivariate signature-only public key cryptosys- tems,'' in Proc. IACR Cryptol. ePrint Arch., Apr. 2001, pp. 1–24.

[82] J.-M. Chen and B.-Y. Yang, ''Tame transformation signatures with topsy-turvy hashes,'' in Proc. IWAP, Taipei, Taiwan, Oct. 2002, pp. 1–8.

[83] L.-C. Wang, Y.-H. Hu, F. Lai, C.-Y. Chou, and B.-Y. Yang, ''Tractable rational map signature,'' in Proc. Int. Workshop Public Key Cryptogr., Les Diablerets, Switzerland, Jan. 2005, pp. 23–36.

[84] Y.-L. Gao, X.-B. Chen, Y.-L. Chen, Y. Sun, X.-X. Niu, and Y.-X. Yang, ''A secure cryptocurrency scheme based on post-quantum blockchain,''

[85] IEEE Access, vol. 6, pp. 27205–27213, 2018. P. Zhang, H. Jiang, Z. Zheng, P. Hu, and Q. Xu, ''A new post- quantum blind signature from lattice assumptions,'' IEEE Access, vol. 6, pp. 27251–27258, 2018.

[86] D. Chaum, ''Blind signatures for untraceable payments,'' in Proc. CRYPTO, Aug. 1982, pp. 199–203.

[87] S. D. Galbraith, C. Petit, and J. Silva, ''Identification protocols and signature schemes based on super singular isogeny problems,'' in Proc. ASIACRYPT, Hong Kong, Dec. 2017, pp. 3–33.

[88] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, ''Sphincs: Practical stateless hash-based signatures,'' in Proc. EUROCRYPT, Sofia, Bulgaria, Apr. 2015, pp. 368–397.

[89] M. Naor and M. Yung, ''Universal one-way hash functions and their cryptographic applications,'' in Proc. 21st Annu. ACM Symp. Theory Comput., Seattle, WA, USA, May 1989, pp. 33–43.