Blockchain Vulnerabilities in Practice

NILS AMIET, Kudelski Security, Switzerland

Blockchains are not invulnerable. There are known vulnerabilities in various blockchain ecosystem components. This field note describes some vulnerabilities observed in smart contracts and node software, their exploitation, and how to avoid them, with a focus on the Ethereum ecosystem.

CCS Concepts: • **Security and privacy** → *Distributed systems security*; *Data anonymization and sanitization*; Denial-of-service attacks;

Additional Key Words and Phrases: Blockchain, security, smart contracts, ethereum

ACM Reference format:
Nils Amiet. 2021. Blockchain Vulnerabilities in Practice. *Digit. Threat.: Res. Pract.* 2, 2, Article 8 (March 2021), 7 pages.
https://doi.org/10.1145/3407230

1 INTRODUCTION

Blockchains are relatively new [13, 36] and there are countless [12, 67] news stories of people losing money through compromises in the components of blockchain ecosystems. Blockchain technologies are not invulnerable and have actually many known vulnerabilities [10, 49, 50], just as with any software. This article describes the most common components of a blockchain ecosystem and the vulnerabilities that they can contain. The greater number of features a blockchain offers, the larger the attack surface becomes.

Ethereum [11], for example, has a virtual machine called the EVM, which executes EVM bytecode. This virtual machine executes smart contracts in a sandboxed environment. The EVM specification defines over 140 different instructions that smart contract programmers can use. EtherVM [63] provides a reference for these instructions. There have been many real-world cases of decentralized applications, also known as DApps [39], leveraging vulnerable smart contracts that resulted in stolen funds [66]. It is also worth noting that vulnerabilities can exist in any component of a blockchain ecosystem, not just smart contracts.

2 BLOCKCHAIN ECOSYSTEM COMPONENTS

A blockchain ecosystem contains multiple components. There is usually a **core blockchain** software that consists of client software such as Go Ethereum [4] or Parity [59] in the case of Ethereum.

Each peer participating in a given blockchain network runs this client software, and is sometimes called a node. Blockchain systems usually contain a **wallet**, which can be implemented in software or hardware. A node

Authors' address: N. Amiet, Kudelski Security, Route de Genève 22-24, 1033 Cheseaux-sur-Lausanne, Switzerland; email: nils.amiet@kudelskisecurity.com.

2576-5337/2021/03-ART8
https://doi.org/10.1145/3407230

and software wallet may be bundled together in the same software. Notable hardware wallets include the Ledger Nano X [33] and the Trezor Model T [62].

Cryptocurrency exchanges are composed of web applications and public REST APIs that can be used to programatically exchange one cryptocurrency for another. A cryptocurrency is a digital currency that is secured by cryptography. Exchanges can use hardware security modules to secure private keys. These exchanges frequently support over a hundred different cryptocurrencies. An exchange requires at least one blockchain node for each supported cryptocurrency. These nodes allow the exchange to perform transactions across various cryptocurrencies, and monitor transactions done on the blockchain to know when it has received cryptocurrencies.

Some blockchains support **smart contracts**: arbitrary code run by multiple nodes of a blockchain network. These contracts are a basic building block for creating decentralized applications. DApps are usually composed of a frontend application, which can be a web application, a desktop application or a mobile application and one or more smart contracts. These smart contracts contain methods that the frontend application can call—by sending a transaction to the contract's address— to perform actions such as reading or writing data to the blockchain.

Last, **e-commerce websites** that want to accept cryptocurrency payments—for example, in Bitcoin [36]—also need a solution that provides this feature. They can either use existing solutions [7, 19] or make their own. In either case, such a solution requires the use of a blockchain node for each cryptocurrency to accept payments properly. All these components present in a blockchain ecosystem can add up and create a large attack surface.

3 KNOWN VULNERABILITIES

There are many known vulnerabilities in blockchain ecosystems. This field note focuses on smart contracts and core blockchain vulnerabilities.

3.1 Core Blockchain Vulnerabilities

3.1.1 Consensus Mechanism Manipulation. Blockchain consensus mechanisms such as proof-of-work or proof-of-stake have been subject to attacks [38]. The 51% attack is a well-known attack on proof-of-work blockchains whereby an attacker who controls the majority of the network's computing resources is able to discard blocks mined by anyone else, giving priority to his own blocks. Note that this attack does not allow an attacker to craft fake transactions or create money out of thin air. It will, however let them perform double spending. In practice, such attacks are extremely costly on blockchain networks with high hashing rates [23], such as Bitcoin, and therefore, quite rare. One possible prevention technique is to perform formal verification [61] of the consensus mechanism.

3.1.2 Underlying Cryptosystem Vulnerabilities. Blockchain wallets usually work with a public and private key pair for signature and are as secure as the underlying cryptosystem they use. The public-key algorithm (ECDSA [32], EdDSA [6], Schnorr [48], ElGamal [24]) used for these keys have known attacks [8, 27, 29, 35, 45, 47] that might be applied in some cases. There are real-world use cases [10, 22]. To reduce the attack surface, it is necessary to use a library that implements the required cryptosystem with side-channel attack protections. One example is the mbedTLS [34] library, which includes side-channel attack protections for deterministic ECDSA signatures.

3.1.3 Improper Blockchain Magic Validation. Some blockchains have multiple forks, such as a main network and one or more test networks. A blockchain's magic value is used to uniquely identify a chain, thus binding transactions to a specific chain. Node software must check whether received transactions have the expected magic value, attributable to the current chain. If there is no such check, then an attacker can replay a transaction originally performed on another chain, thus creating transactions meant for another chain.

3.1.4 Improper Transaction Nonce Validation. Each transaction must be unique within a given blockchain. A transaction nonce is used by node implementations to enforce uniqueness. Poor node implementations might allow transactions to be replayed on the same chain. Such a vulnerable implementation may be exploited by an attacker that receives a transaction of amount N, which can be replayed over and over until the source wallet does not have enough funds to perform the transaction anymore. A mitigation is to verify the uniqueness of all received transactions.

3.1.5 Denial of Service. Proof-of-work blockchains with a block target [60] that automatically adjusts, may be vulnerable to a denial of service attack. Indeed, if no minimum target is defined, an uncaught floating-point underflow may occur and the block target may be rounded to zero, thus making new blocks impossible to mine and rendering the blockchain useless. Some DApps may become so popular [65] that congestion may happen on the underlying blockchain. One way to avoid network congestion, is to use a smart contract platform that features high throughput [68].

3.1.6 Public-key and Address Mismatch. A blockchain wallet's public key can usually be derived from the wallet's address. However, some implementations [3] truncate the public key to derive the address. If addresses are not bound to a specific key pair, then this can become a problem, because multiple key pairs exist with the same wallet address. In this case, it may be possible to brute force and find another key pair that controls a target wallet in a reasonable amount of time depending on the address length. To prevent such attacks, one should ensure that each unique wallet address is bound to a single key-pair.

3.2 Smart Contract Vulnerabilities

As of July 2020, Ethereum is still considered the most popular DApp platform [56]. Most DApps use Ethereum and the platform has the highest number of daily active users and of smart contracts. After Ethereum, EOSIO [25] and Steem [58] are the most widely used. Ethereum's popularity most likely comes from the fact that it was the first blockchain to offer quasi-Turing complete (except for the gas limit) smart contracts with a low enough latency (sustained 13 s average block time) acceptable for most decentralized applications. Ethereum smart contracts are usually written in a high-level language such as Solidity [55] or Vyper [64]. Both languages are compiled to EVM bytecode. There are multiple other blockchains [1, 14, 25, 31, 37, 57] supporting smart contracts nowadays. The rest of this section enumerates known smart contract vulnerabilities that may appear in any of the aforementioned smart contract platforms, but uses Ethereum to provide examples [20, 28, 54].

3.2.1 Reentrancy. A reentrancy vulnerability [12] can be exploited when a contract function **F** whose purpose is to withdraw funds, synchronously calls another untrusted contract's default function **D**. Indeed, D can call F again before F updates its state. If F does not update its state before performing the external call to D, then it may be vulnerable (Listing 1). Indeed, a malicious external contract may call the withdraw() function, which will call the malicious contract's default function, which can call the withdraw() function again before the balance is updated, thus withdrawing more funds than it should be able to.

```
1 function withdraw(uint x) {
2   require(balances[msg.sender] >= x);
3   msg.sender.call.value(x)();
4   balances[msg.sender] -= x;
5 }
```

Listing 1. Reentrancy vulnerable withdraw() function. Lines 3 and 4 should be swapped to fix the vulnerability.

In 2016, a smart contract part of the DAO [12] was found to be vulnerable to reentrancy and was subject to an attack that led to the loss of 3.6 million ETH or about $50 million USD at the time. This very event caused an

Ethereum hard fork. The original Ethereum chain was renamed to Ethereum Classic [18], and the new fork took the original name of Ethereum.

3.2.2 Arithmetic Issues. The Solidity smart contract language does not catch integer overflows by default. If uncaught, then overflows can lead to unexpected behavior (Listing 2). Unsigned integers are represented with 256 bits in Solidity. Therefore, an arithmetic operation on an unsigned integer that causes the result to be greater than $2^{256} - 1$ or less than 0 may be exploited [5, 42, 43].

```
function withdraw(uint x) {
  require(balances[msg.sender] - x > 0);
  msg.sender.transfer(x);
  balances[msg.sender] -= x;
}
```

Listing 2. Integer overflow vulnerable withdraw() function. What happens if x is large?

One solution to this problem is to use secure functions provided by an external smart contract library such as OpenZeppelin [40]'s SafeMath functions or to use a language that has a built-in protection against overflows such as Vyper [64].

3.2.3 Unprotected SELFDESTRUCT. The SELFDESTRUCT EVM instruction makes a smart contract unusable and sends the contract's balance to the address given in parameter. If a contract has a self-destruction functionality, then it must be protected with care [67], and it must be ensured that only authorized users can call the code. As a general rule, the use of *selfdestruct* should be avoided if possible. If it cannot be avoided, then it must be used with extreme care.

3.2.4 Visibility Issues. It is a best practice to explicitly mark function visibility for all functions and variables. The default visibility is public for functions in Solidity. If a function's visibility is not explicitly marked, then a developer can easily conclude that the function is private while it is actually public, causing unexpected behavior such as letting an attacker call supposedly private code.

Similarly, any data that is written to the EVM storage area is visible by anyone, because it is stored on the blockchain. Any call to another contract, including function arguments, is publicly visible as well, because it creates a transaction. Secrets should not be stored in clear text in the EVM storage.

3.2.5 Weak Randomness. Generating random numbers in smart contracts is a hard problem [30, 44]. Smart contract developers who need random numbers may be tempted to use predictable chain data as a source of randomness. Such chain data include the block number, the block hash and the block timestamp. All of these values can actually be manipulated by block miners and should not be used for generating random numbers. The SmartBillions [53] contract is a famous example of poor block hash usage, which allowed attackers to exploit the contract and withdraw all the lottery money. One possible solution is to use a secure random number generator for smart contracts such as RANDAO [44] or RBGC [16].

3.2.6 Transaction Order Dependence (Front Running). Transaction order dependence may lead to unfair remuneration [9, 52] in the case of smart contracts. Imagine that a smart contract implements a quiz that asks players to solve a problem. The first player that sends the correct solution to the problem can claim the prize (the contract's balance). Now let us say Alice worked day and night for a month and finally finds the solution to the problem. She makes a transaction to send her solution. An attacker sees Alice's solution in the transaction pool and immediately sends the same solution but with higher transaction fees. The attacker's transaction is selected first by miners because of the higher fees paid by the attacker, and that transaction gets inserted in a block before Alice's. The attacker is, therefore, the first to solve the problem and can claim the prize instead of

Alice. To prevent this attack, it is possible to use a commit-reveal commitment scheme [26] that allows players to securely disclose the solution to the problem.

3.3 Education and Further Reading

There are many blockchain security education tools. FumbleChain [51] contains tutorials and challenges based on a purposefully vulnerable blockchain written in Python, allowing developers to learn about base blockchain security. Ethernaut [41] is a wargame focused on Ethereum smart contract vulnerabilities in Solidity. There are several good articles discussing blockchain security [2, 17, 46] as well.

4 CONCLUSION

Selected vulnerabilities associated with core blockchain and smart contracts were detailed. The way these vulnerabilities can be exploited and mitigation techniques to prevent such attacks were presented. Blockchains are not invulnerable, and one should use a mix of dedicated tools [15, 21] for smart contracts and general software analysis tools, such as static code analyzers for core blockchain, to automatically detect simple issues. Good practices, such as software testing, also apply to blockchain development. Thorough code reviews or third party security audits should be performed to maximize detection of more complex vulnerabilities.

REFERENCES

[1] Algorand. 2018. Algorand. Retrieved from https://www.algorand.com/.

[2] Ayman Alkhalifah, Alex Ng, A. S. M. Kayes, Mohammad Chowdhury, Mamoun Alazab, and Paul Watters. 2019. A Taxonomy of Blockchain Threats and Vulnerabilities. DOI: https://doi.org/10.20944/preprints201909.0117.v1

[3] J. P. Aumasson. 2018. Blockchains: How to Steal Millions in 2**64 Operations. Retrieved from https://research.kudelskisecurity.com/2018/01/16/blockchains-how-to-steal-millions-in-264-operations/.

[4] Various authors. 2013. Go Ethereum—Official Go implementation of the Ethereum protocol. Retrieved from Retrieved from https://github.com/ethereum/go-ethereum.

[5] Eric Banisadr. 2018. How 800k Evaporated from the PoWH Coin Ponzi Scheme Overnight. Retrieved from https://medium.com/@ebanisadr/how-800k-evaporated-from-the-powh-coin-ponzi-scheme-overnight-1b025c33b530?.

[6] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2011. High-speed high-security signatures. *Cryptology ePrint Archive, Report 2011/368*. Retrieved from https://eprint.iacr.org/2011/368.

[7] Bitpay. 2016. Bitpay. Retrieved from https://bitpay.com/.

[8] Daniel Bleichenbacher. 2000. Generating ElGamal signatures without knowing the secret key. https://crypto.ethz.ch/publications/files/Bleich96.pdf.

[9] Ivan Bogatyy. 2017. Implementing Ethereum trading front-runs on the Bancor exchange in Python. Retrieved from https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798.

[10] Joachim Breitner and Nadia Heninger. 2019. Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies. *Cryptology ePrint Archive, Report 2019/023*. Retrieved from https://eprint.iacr.org/2019/023.

[11] Vitalik Buterin. 2015. Ethereum. Retrieved from https://ethereum.org/.

[12] Vitalik Buterin. 2016. Critical Update Re: DAO Vulnerability. Retrieved from https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/.

[13] Vitalik Buterin. 2019. Hard Problems in Cryptocurrency: Five Years Later. Retrieved from https://vitalik.ca/general/2019/11/22/progress.html.

[14] Cardano. 2014. Cardano. Retrieved from https://www.cardano.org/en/home/.

[15] ChainSecurity. 2018. Chaincode Scanner. Retrieved from https://chaincode.chainsecurity.com/.

[16] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. 2019. Probabilistic smart contracts: Secure randomness on the blockchain. In *Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC'19)*. DOI: https://doi.org/10.1109/bloc.2019.8751326

[17] Vincent Chia, Pieter Hartel, Qingze Hum, Sebastian Ma, Georgios Piliouras, Daniël Reijsbergen, Mark Van Staalduinen, and Pawel Szalachowski. 2018. Rethinking blockchain security: Position paper. In *Proceedings of the IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData'18)*. IEEE, 1273–1280.

[18] Ethereum classic. 2016. Ethereum Classic. Retrieved from https://ethereumclassic.org/roadmap.

[19] Coinbase. 2018. Coinbase Commerce. Retrieved from https://commerce.coinbase.com/.

[20] ConsenSys. 2016. Known Attacks. Retrieved from https://consensys.github.io/smart-contract-best-practices/known_attacks/.
[21] ConsenSys. 2016. Smart contract security tools. Retrieved from https://consensys.github.io/smart-contract-best-practices/security_tools/.
[22] Nicolas T. Courtois, Filippo Valsorda, and Pinar Emirdag. 2014. Private Key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. https://eprint.iacr.org/2014/848.
[23] Crypo. 2018. PoW 51% Attack Cost. Retrieved from https://www.crypto51.app/.
[24] T. ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Info. Theory* 31, 4 (July 1985), 469–472. DOI: https://doi.org/10.1109/TIT.1985.1057074
[25] EOS. 2018. EOSIO. Retrieved from https://eos.io/.
[26] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. SoK: Transparent Dishonesty: front-running attacks on Blockchain. https://arxiv.org/abs/1902.05164.
[27] P. Fouque, D. Masgana, and F. Valette. 2009. Fault attack on Schnorr-based identification and signature schemes. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'09)*. 32–38. DOI: https://doi.org/10.1109/FDTC.2009.36
[28] NCC Group. 2018. Decentralized Application Security Project. Retrieved from https://dasp.co/.
[29] R. A. Haraty, H. Otrok, and A. Nasser Kassar. 2005. Attacking ElGamal-based cryptographic algorithms using Pollard's rho algorithm. In *Proceedings of the 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.*91. DOI: https://doi.org/10.1109/AICCSA.2005.1387082
[30] Tjaden Hess and Niklas Feurstein. 2016. How can I securely generate a random number in my smart contract? Retrieved from https://ethereum.stackexchange.com/questions/191/how-can-i-securely-generate-a-random-number-in-my-smart-contract/207#207.
[31] Hyperledger. 2016. Hyperledger Fabric. Retrieved from https://www.hyperledger.org/projects/fabric.
[32] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Info. Secur.* 1, 1 (Aug. 2001), 36–63. DOI: https://doi.org/10.1007/s102070100002
[33] Ledger. 2019. Ledger Nano X. Retrieved from Retrieved from https://shop.ledger.com/pages/ledger-nano-x.
[34] ARM Limited. 2009. mbed TLS. Retrieved from https://tls.mbed.org/.
[35] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. 2015. On the Security of the Schnorr Signature Scheme and DSA against Related-Key Attacks. *Cryptology ePrint Archive, Report 2015/1135*. Retrieved from https://eprint.iacr.org/2015/1135.
[36] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list. Retrieved from https://metzdowd.com.
[37] Neo. 2014. Neo. Retrieved from https://neo.org/.
[38] ocminer. 2018. Network Attack on XVG / VERGE. Retrieved from https://bitcointalk.org/index.php?topic=3256693.0.
[39] State of the DApps. 2014. State of the DApps. Retrieved from https://www.stateofthedapps.com/.
[40] OpenZeppelin. 2016. OpenZeppelin. Retrieved from https://github.com/OpenZeppelin/openzeppelin-contracts.
[41] OpenZeppelin. 2017. Ethernaut. Retrieved from https://ethernaut.openzeppelin.com/.
[42] PeckShield. 2018. New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018â10299). Retrieved from https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536.
[43] pirapira. 2016. Exception on overflow. Retrieved from https://github.com/ethereum/solidity/issues/796#issuecomment-253578925.
[44] RANDAO. 2015. RANDAO: A DAO working as RNG of Ethereum. Retrieved from https://github.com/randao/randao.
[45] Y. Romailler and S. Pelissier. 2017. Practical fault attack against the Ed25519 and EdDSA signature schemes. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'17)*. 17–24. DOI: https://doi.org/10.1109/FDTC.2017.12
[46] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and Aziz Mohaisen. 2019. Exploring the attack surface of blockchain: A systematic overview. Retrieved from https://arXiv:1904.03487.
[47] Jorn-Marc Schmidt and Marcel Medwed. 2009. A fault attack on ECDSA. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'09)*. 93–99. DOI: https://doi.org/10.1109/FDTC.2009.38
[48] Claus Schnorr. 1991. Efficient signature generation by smart cards. *J. Cryptol.* 4 (Jan. 1991), 161–174. DOI: https://doi.org/10.1007/BF00196725
[49] Eric Schorn. 2018. Smart Contract (in)Security—Bad Arithmetic. Retrieved from https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2018/november/smart-contract-insecurity-bad-arithmetic/.
[50] Eric Schorn. 2018. Smart Contract (in)Security—Fat Fingers. Retrieved from https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2018/december/smart-contract-insecurity-fat-fingers/.
[51] Kudelski Security. 2019. FumbleChain: A Purposefully Vulnerable Blockchain. Retrieved from https://github.com/kudelskisecurity/fumblechain.
[52] Emin Gun Sirer and Phil Daian. 2017. Bancor Is Flawed. Retrieved from http://hackingdistributed.com/2017/06/19/bancor-is-flawed/.
[53] SmartBillions. 2017. SmartBillions—World's first multi-billion-dollar decentralized global blockchain lottery. Retrieved from https://twitter.com/smartbns.
[54] SmartContractSecurity. 2018. Smart Contract Weakness Classification and Test Cases. Retrieved from https://swcregistry.io/.

[55] Solidity. 2014. Solidity. Retrieved from https://solidity.readthedocs.io/.
[56] DApp Statistics. 2014. DApps Statistics. Retrieved from https://www.stateofthedapps.com/stats/.
[57] Steem. 2016. Steam—Write a smart contract. Retrieved from https://github.com/harpagon210/steemsmartcontracts-wiki/blob/master/Write-a-Smart-Contract.md.
[58] Steem. 2016. Steem. Retrieved from https://steem.com/.
[59] Parity Technologies. 2015. Parity—The fast, light, and robust EVM and WASM client. Retrieved from https://github.com/paritytech/parity-ethereum.
[60] Theymos. 2010. Block target and difficulty. Retrieved from https://en.bitcoin.it/wiki/Target.
[61] Pierre Tholoniat and Vincent Gramoli. 2019. Formal Verification of Blockchain Byzantine Fault Tolerance. Retrieved from https://arxiv:1909.07453.
[62] Trezor. 2018. Trezor ModelT. Retrieved from https://shop.trezor.io/product/trezor-model-t.
[63] Unknown. 2018. EtherVM - Ethereum Virtual Machine Opcodes. Retrieved from https://ethervm.io/.
[64] Vyper. 2016. Vyper. Retrieved from https://vyper.readthedocs.io/en/latest/index.html.
[65] Joon Ian Wong. 2017. The ethereum network is getting jammed up because people are rushing to buy cartoon cats on its blockchain. Retrieved from https://qz.com/1145833/cryptokitties-is-causing-ethereum-network-congestion/.
[66] Joseph Young. 2018. Another ICO Hacked: KICKICO Loses 8 Million After Smart Contract Breach. Retrieved from https://finance.yahoo.com/news/another-ico-hacked-kickico-loses-120331205.html.
[67] Michael Yuan. 2017. I accidentally killed it (and evaporated 300 million). Retrieved from https://medium.com/cybermiles/i-accidentally-killed-it-and-evaporated-300-million-6b975dc1f76b.
[68] Zilliqa. 2019. Zilliqa—High-throughput smart contract processing platform. Retrieved from https://zilliqa.com.

Received December 2019; revised January 2020; accepted June 2020