Abstract

Abstract —

Smart contracts have emerged as a technology with the ability to transform the enforcement of rights and rules in the digital world. This has raised questions on the security and privacy of smart contracts. The lack of development tools and training has led to many vulnerabilities in smart contracts. The development of models to formally prove the correctness of these frameworks is under exploration. Modern cryptographic primitives are implemented to achieve privacy in public blockchains. Anonymous decentralized frameworks raise concerns and open new challenges for the future. The present literature review is based on the Ethereum blockchain, the most widely adopted platform.

Key words — Smart contracts, blockchains, Ethereum, security, privacy.

Contents

1 Introduction

Summary

The research review starts with an **Introduction**. *Summary* presents a high-level description on the aims of this report and gives a high-level overview of the smart contracts development paradigm. *Technical background* serves as a primer on smart contracts and distributed ledgers for readers who do not have the necessary background.

The introduction is followed by an in-depth analysis of the **Security landscape** of smart contracts in Ethereum. *The smart contracts programming paradigm* reflects the misconceptions and main vulnerabilities in development of smart contracts in Ethereum. *Improving the security standards* discusses efforts to achieve more secure development frameworks, ranging from building more adequate programming languages to improvement of testing tools.

Privacy on the blockchain starts by discussing the apparent opposition of public and private in *Are openness and privacy compatible?*. This section overviews crytographic primitives used to achieve privacy on the blockchain. *Provably secure and private models* suggest solutions to these challenges in the form of formal models. The last section discusses *Ethical concerns in anonymous systems*. It introduces negative or unknown consequences of private and decentralized systems.

Conclusions condenses the state of the art of smart contracts in distributed ledgers and reviews the open challenges in the roadmap to real world adoption.

Technical background

A contract is an agreement on a set of premises between two or more parties which formalizes their relationship. The main historical usage of contracts is to establish business relationships, but their utility can range from the declaration of ownership of assets to the establishment of a marriage [1]. The validity of these agreements is commonly enforced by the government of each jurisdiction. However, with the digital revolution, a new world of possibilities was already envisioned by Szabo in 1996 [1], when he introduced the idea of smart contracts. These digital protocols would be both declared and enforced by code, performing as *trusted third parties* [2]. A rudimentary example on a conceptual level would be a vending machine, which delivers a snack to a customer when a coin is inserted, following an established protocol [1].

The concept of smart contracts became a reality when Satoshi Nakamoto published the Bitcoin White Paper in 2009 [3]. Following early attempts to develop an electronic currency or e-cash [4,5], Bitcoin offered a solution to the double spending of the currency without the need for a trusted central authority. This open challenge in distributed systems was overcome with the implementation of an innovative data structure: the blockchain. A blockchain, also known as a distributed ledger, is an append-only log which is maintained and extended by a peer-to-peer network of distrustful nodes, called *miners*[3]. It can be seen as a *hash-chain*, a linked-list of blocks in which each node of the list contains the hash of the previous one. In essence, Bitcoin resolved the double-spending obstacle combining a cryptographically secured database with a distributed consensus protocol which extends the ledger [3]. Although Bitcoin was envisioned as a peer-to-peer payment network, the agreement of the nodes on a global state of the blockchain could also serve as the *trustless enforcer* (a provider of availability and correctness) of smart contracts deployed in the ledger [6].

To overcome the niche applications to finance of Bitcoin, Ethereum was created as a general-purpose transaction-based state machine, commonly labeled as a *world-computer*, as all the code of its blokchain would run redundantly in every node of the network [7]. A contract (or a transaction) is enforced when it is deployed to the ledger, after all the nodes have reached consensus on the global state of the platform. Ethereum has emerged as the most prominent smart contracts platform, given its Turing completeness, the theoretical possibility to solve any computational problem. To surpass the naive denial-of-service vulnerability of the network while permitting the usage of loops and recursion, Ethereum introduces the concept of *gas*, which can be thought of as the cost of running each operation in the blockchain. At the core of Ethereum resides the Ethereum Virtual Machine (EVM), which serves as a runtime environment for programs written in higher-level languages (prior compilation to Ethereum byte-code) for execution on the global network of computers [7].

Ethereum is a *permissionless* blockchain, meaning anyone can become a user by creating a *key-pair*, with each user being identified by his public key [8]. Each contract is also linked to a unique address in the blockchain and any user can interact with it once it is deployed. To illustrate this concept, the code snippet in Fig 1.1 shows a naive implementation of a domains registry handled by a contract, like the DNS in the World Wide Web [9]. The code is written in Solidity, the most used language for development on Ethereum [11]:

```
1  contract DomainRegistry {
2
3   mapping (string => string) domains;
4
5   function registerDomain (string name,string value) returns boolean {
6    if (domains[name] == 0) {
7     domains[name] = value;
8     return true;
9    }
10   return false;
11  }
12
13  function getDomain (string name) returns string {
14   return domains [name];
15  }
16 }
```

Figure 1.1 :A simple domain registry written in Solidity as in [9].

As we can see in Figure 1.1, a smart contract in Ethereum is comprised of data fields and functions which any user can call either to alter or retrieve the data. In the case of the domain registry, the first function allows users to register a domain and the second one to retrieve an already stored domain. Additionally, contracts can store *ether*, the cryptocurrency of the network. Users can invoke a contract by sending ether to the contract address. In the same way, any user can deploy a contract to the network by paying the relevant *execution fees* in gas [7].

2

Security landscape

Before we explore the security of smart contracts in Ethereum, we will assume the integrity and correctness of the Ethereum network. Theoretically, a distributed ledger will be of trust, if at least 51% of the nodes are honest [1]. Several studies have questioned the validity of the threshold, pointing out that if a subset of miners cooperate and behave with an unexpected protocol, the fairness of the blockchain could be compromised [10].

The smart contracts programming paradigm

As researchers from the University of Maryland have shown, while programming smart contracts may resemble common software development, the conceptual differences can be critical [2]. Even if the code of the contract is free of bugs, a correct alignment of incentives is necessary (i.e. how rational users are incentivized to interact with the contract). In any interaction between users and contracts, there is an underlying transfer of value in the form of *Ether* (in the case of Ethereum). Therefore, developers are encouraged to take in account the game-theoretic analysis on how rational users will interact with the program. Delmolino et al. have shown the lack of training and awareness in adding this economic perspective to program software [2]. Smart contracts are prone to vulnerabilities when developers fail to align the intended logic of the contract with the incentives of potential users [11,12,13].

The surface of attack of smart contracts is broad. The fact that they are designed for handling digital assets makes them easily monetizable targets [12]. If a rogue player finds an exploit by supplying unexpected input, they may attain huge profits, as the valuation of cryptocurrencies are in the order of the hundreds of billions of dollars [14]. The best example of such an exploit is the so-called *DAO hack*. Funded by large quantities of *Ether*, the DAO (Decentralized Autonomous Organization) was envisioned as a digital organization with no rulers, where *the only law was the code* [15]. However, a hacker used the recursion capabilities of Ethereum to exploit a function of the DAO vulnerable to reentrancy [16]. As has been stated, the integrity of the blockchain is backed by its non-reversibility, being by definition an append-only ledger. Therefore, when a vulnerability is found, the *immutability* of the ledger makes it permanent. The lack of bugs in smart contracts becomes critical, compared to the periodic fixes and patches present in common software practices. When a bug is found, developers are forced to add complexity to the code, in order to raffle the vulnerability [12]. It should be mentioned that in the case of the DAO, given the magnitude of the exploit, the Ethereum blockchain forked [17] to a state previous to the hack, raising a debate which will be later discussed.

The other major cause of vulnerabilities in smart contracts development comes from the misconception of the inner workings of blockchains. The transactions on the network are stored when a consensus by the miners is reached. However, taking into account that the ledger is a distributed system, the ordering on which these transactions are appended may vary owing to the latency of the network and the consensus algorithm itself [4,8]. Contracts which reach different states given a change in the order of transactions suffer from so-called Transaction order Dependent (TOD) bugs [13]. Additionally, it has been shown that the openness of blockchains as public ledgers is not inherently natural for developers [2]. Every transaction on the ledger can be seen by anyone. Contracts that fail to use cryptographic techniques to encrypt sensitive data are vulnerable to be exploited by any malicious user. Examples like card games or *Rock,Paper, Scissor* style games on smart contracts illustrate the need of secret commitments to behave as expected [2,12].

Improving the security standards

As described in the previous section, one of the main security issues in the development of smart contracts comes from the semantic gap between the intention and the implementation of smart contracts creators [18]. Therefore, one of the main causes of the high degree of vulnerable code is the misuse of Solidity. Owing to its similarities with well-known languages such as JavaScript [11] or Python [19], developers usually inherit software design patterns of the later programming languages. To proactively overcome these error-prone behaviors, more constraint languages are being tested [20] by pruning the flexibility and complexity of Solidity and its Turing completeness (i.e. no loops or recursive calling). Heavily restricted programming languages with extensive type systems are explored to reduce unexpected input from rogue users. These efforts question if an imperative programming language like Solidity is adequate to formalize contracts, com-

monly associated with declarative formal languages [10]. Other initiatives are pursuing more radical approaches to find programming languages which ease the development of contracts. An example of this trend is the development of visual programming languages that build physical analogies (e.g. vending machines or water pipes) to emulate the mechanical design of a contract [21]. On the other hand, different alternatives to educate students on the subject are starting to flourish [2]. Academic papers [12] and open-source documentation [11,13,22] have been also developed to enrich didactic resources available to developers.

Different tools have been launched to better predict how the smart contract will behave. They provide virtual frameworks to thoroughly test the code before deployment on the real network [23]. In particular, Luu et al. have built Oyente [7], a symbolic execution tool especially designed for Ethereum. Oyente enhances the operational semantics of Ethereum and has discovered most of the known reported buggy contracts (by May 2016). Another parallel approach is the development of frameworks which translate Solidity to functional programming languages designed for formal verification techniques [24]. In a similar fashion, Kosba et al. proposed a model to formally prove the correctness of contracts before deployment [25]. What these initiatives encapsulate is the need for more robust testing and correctness verification methods, given the impossible nature of fixing any issues after deployment.

3

Privacy on the blockchain

This section explores the challenges of building private systems which rely on open distributed ledgers. It starts with an overview of the main ideas and techniques to achieve privacy on blockchains and later focuses on the possible applications to Ethereum and smart contracts development. After that, a brief discussion about the ethical concerns in this new paradigm will be presented.

Are openness and privacy compatible?

Public blockchains like Bitcoin or Ethereum are said to be *pseudonymous* [4]. Every user and contract is linked to a public address (i.e a public key) which serves as an identifier in the system [4] In contrast, an anonymous system lacks any type of identity for the users. In spite of that, Reid and Harrigan [26] have showed that the *linkability* between the public address and the real-world users can be unmasked through a graph analysis of the transactions publicly stored in the ledger. The accessibility of the log of all the transactions which are stored in the blockchain represents a difficult obstacle to obtain anonymous interactions. This weakens the *fungibility* (one unit of Ether being exactly as valuable as any other, regardless of past transactions or owners) of cryptocurrencies like Ether or Bitcoin [26].

One of the most common solutions to hide the footprints of users in technological networks is using *mix-nets*, which hide one particular transaction in a bigger pool of uncorrelated transactions [27]. However, this approach is more suited to transaction-based systems such as Bitcoin. Although these techniques can anonymize the exchange of *Ether*, the concept of privacy in a state-based system such as Ethereum is broader. How

can a user set a value in the storage of a smart contract as a private commitment? In this case, we are trying to ensure the privacy of both the interaction and the data stored in the ledger, not the identity of the user who performs the transaction. The answer to this question is a complex cryptographic primitive known as *zero-knowledge* proof (zk-proof) [28]. At a higher level, a zk-proof allows a party to prove possession of a determined *knowledge* without actually revealing it. It should be noted that, owing to the redundancy of the code being executed on the blockchain, a computationally efficient variant of zk-proofs is actually used, called *zk-snarks*. The implementation of these techniques has been widely explored in other blockchains like Zerocoin, built on top of the Bitcoin network [29,30]. Recently, Ethereum has been upgraded and allows the use of *zk-snarks* on smart contracts on its platform. These advances open a new array of applications leveraged by their privacy possibilities [31].

Provably secure and private models

Academia has made efforts to formally prove the security of blockchain systems. The objective is to ensure the security of the whole framework using formal models [32]. Following Kosba et al. have developed a provably secure and private model of a smart contracts development framework [25]. This model follows the Universal Composability framework proposed by R. Canetti, typical standard on the analysis of cryptographic protocols [33]. It relies on special-purpose security hardware [34], which can ensure both privacy and security in remote computation environments. While there is not yet a practical implementation of these approaches, these initiatives are grounding the foundations of provably secure and private smart contracts.

Ethical concerns in anonymous systems

Many possible real-world applications can greatly benefit from the capability of using private smart contracts (e.g. health records). However, fully private systems can also increase malicious behavior. The pseudonymity of Bitcoin have already raised concerns in different sectors of society. Cryptocurrencies have already been used in money laundering and cyber-crime ventures, with infamous examples such as Silk Road, an online black market hosted in the darknet [35]. Juels et al. [36] have studied the possible cases of misuse that will emerge with completely anonymous and private smart contracts.

The decentralized nature of blockchains leads to the problem of accountability in smart contract platforms [37]. The trust is deposited on distributed ledgers as the enforcers of contracts, under the premise of *the code is law* [15]. When a user interacts with the contract in a way permitted by the code (as in the DAO hack), should it be considered an exploit? The governance and responsibility distribution on distributed ledgers is becoming a topic of debate nowadays [37]. As with any other technology, the management and limitations of its potential positive or negative effects will remain an ethical debate.

4

Conclusions

The blockchain ecosystem and the development of decentralized smart contracts frameworks is at its infancy. Nevertheless, in recent years a burst of innovative research efforts and ventures have emerged. The secure development of smart contracts is still a challenge, with vulnerabilities being founded in almost the half of the programs deployed in popular frameworks such as Ethereum [7]. The semantic gap between programmers and the blockchain paradigm has been identified as the main cause of vulnerabilities. As a consequence, different groups are exploring tools to test and verify the correctness of the code, improving the security standards of the ecosystem. Additionally, new programming languages specifically defined for this purpose, with more constraints than currently adopted languages (i.e. Solidity) are currently being built. It is questionable if the Turing completeness of Ethereum is adequate to encode the management of assets in the digital world. The security of these systems is critical and the development of smart contracts has been compared with access-control systems [38], so the viability of such a flexible framework such as Ethereum remains unknown.

Another key component in the evolution and maturity of distributed ledgers is privacy. Achieving private frameworks presents a real challenge owing to its conceptual opposition to the openness of public blockchains. Cryptographic tools and protocols already used by networks are being implemented to *unlink* the real world users with their public addresses. Furthermore, there are efforts to create provable models which will guarantee a secure and private environment for the development of smart contracts. Fully anonymous systems may facilitate malicious behaviors, raising ethical concerns about their viability. The ecosystem faces many future challenges, from the legal implications of this decentralized paradigm to the deployment of reliable smart contracts platforms, which will need to be solved for a high-scale real world adoption of the technology.

Bibliography

[1] Szabo,N. Smart Contracts: Building Blocks for Digital Markets. 1996. *http://www.alamut.com/subj/economics/nick_szabo/smartContracts.html*

[2] Delmolino,K. et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security.* Springer Berlin Heidelberg, 2016. p. 79-94.

[3] Nakamoto,S. Bitcoin: A peer-to-peer electronic cash system. 2008. *https://bitcoin.org/bitcoin.pdf*

[4] Chaum,D et al. Untraceable electronic cash. In *Proceedings on Advances in cryptology*(pp. 319-327). Springer-Verlag New York, Inc..

[5] Back,A. Hashcash-a denial of service counter-measure. 2002. *http://www.hashcash.org/papers/hashcash.pdf*

[6] Luu,L. et al. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2016. p. 254-269.

[7] Wood,G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014, vol 151.

[8] Hellman,M.L. An overview of public key cryptography. In *IEEE Communications Magazine*, 2002. 40(5), 42-49.

[9] Pettersson,J. and Edström,R. Safer smart contracts through type-driven development. 2016. *https://allquantor.at/blockchainbib/pdf/pettersson2016safer.pdf*

[10] Eyal,I. and Sirer,E.G. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security.* Springer, Berlin, Heidelberg, 2014. p. 436-454.

[11] Solidity Documentation. *https://solidity.readthedocs.io/en/develop/.*

[12] Atzei,N. et al. Survey of Attacks on Ethereum Smart Contracts (SoK). In *International Conference on Principles of Security and Trust* (pp. 164-186). Springer, Berlin, Heidelberg. ISO 690

[13] Ethereum Smart Contract Best Practices, *Consensys* *https://consensys.github.io/smart-contract-best-practices/known_attacks/.*

[14] Cryptocurrency Market Capitalizations *https://coinmarketcap.com/*

[15] Jentzsch,C. Decentralized autonomous organization to automate governance. *https://download.slock.it/public/DAO/WhitePaper.pdf*

[16] Daian,P. Analysis of the DAO exploit. *http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/*

[17] Buterin,V. Hard Fork Completed *https://blog.ethereum.org/2016/07/20/hard-fork-completed/*

[18] Buterin, V. Thinking about smart contract security *https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/*

[19] Serpent Ethereum programming language. *https://github.com/ethereum/wiki/wiki/Serpent.*

[20] Viper experimental programming language. *https://github.com/ethereum/viper.*

[21] A mechanical smart contract language. *https://medium.com/@chriseth/babbage-a-mechanical-smart-contract-language-5c8329ec5a0e.*

[22] OpenZeppelin, a framework to build secure smart contracts on Ethereum. *https://github.com/OpenZeppelin/zeppelin-solidity*

[23] Remix Solidity IDE. *https://remix.ethereum.org*

[24] Bhargavanan,K. et al. Formal verification of smart contracts. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security-PLAS'16.* 2016. p. 91-96.

[25] Kosba,A. et al. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on* IEEE, 2016. p. 839-858.

[26] Reid,F. and Harrigan,M. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks.* Springer New York, 2013. p. 197-223.

[27] Bonneau,J. et al. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security.* (pp. 486-504) Springer, Berlin, Heidelberg.

[28] Goldwasser,S. et al. The knowledge complexity of interactive proof systems. In *SIAM Journal on computing, 18(1)* 1989. 1(2), 186-208.

[29] Miers,I. et al. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, IEEE, 2013. p. 397-411.

[30] Sasson,E.B. et al. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on.*IEEE, 2014. p. 459-474.

[31] Wilcox,Z. Ethereum Adoption of zk-SNARK Technology. *https://z.cash/blog/ethereum-snarks.html*

[32] Garay,J.A. et al. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT (2)* 2015. p. 281-310.

[33] Canetti,R. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*IEEE, 2001. p. 136-145.

[34] Costan,V. et al. Intel SGX Explained. *IACR Cryptology ePrint Archive,* 2016, vol. 2016, p. 86.

[35] The Untold Story of Silk Road. *Wired,2015. https://www.wired.com/2015/04/silk-road-1*

[36] Juels,A. et al. The ring of Gyges: Investigating the future of criminal smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2016. p. 283-295.

[37] Ehrsam,F. Blockchain Governance: Programming Our Future. *https://medium.com/@FEhrsam/blockchain-governance-programming-our-future-c3bfe30f2d74*

[38] Arcieri,T. Ethereum's Hindenburg moment: The DAO disaster and smart contract security takeaways. *https://tonyarcieri.com/a-tale-of-two-cryptocurrencies*