Rethinking Blockchain Security: Position Paper (preprint)

Vincent Chia[*], Pieter Hartel[†], Qingze Hum[†], Sebastian Ma[*], Georgios Piliouras[†]
Daniël Reijsbergen[†], Mark van Staalduinen[*], Pawel Szalachowski[†]

[*]The Netherlands Organisation for Applied Scientific Research (TNO)
{*vincent.chia, sebastian.ma, mark.vanstaalduinen*}@tno.nl
[†]Singapore University of Technology and Design
{*pieter_hartel, georgios, daniel_reijsbergen, pawel*}@sutd.edu.sg
qingze_hum@mymail.sutd.edu.sg

Abstract—**Blockchain technology has become almost as famous for incidents involving security breaches as for its innovative potential. We shed light on the prevalence and nature of these incidents through a database structured using the STIX format. Apart from OPSEC-related incidents, we find that the nature of many incidents is specific to blockchain technology. Two categories stand out: smart contracts, and techno-economic protocol incentives. For smart contracts, we propose to use recent advances in software testing to find flaws before deployment. For protocols, we propose the PRESTO framework that allows us to compare different protocols within a five-dimensional framework.**

Index Terms—**blockchain, security, smart contracts**

I. INTRODUCTION

The idea of the blockchain — an open ledger maintained and extended by a peer-to-peer network — is shaping up to become a foundational technology underpinning not just cryptocurrencies but also a diverse range of other applications [1]–[4]. However, its novelty means that its surrounding security infrastructure has some way to go to reach the level of more established digital services such as online banking. Combined with the increased valuation of tokens, this has made cryptocurrency platforms a popular target for attacks. This is witnessed by several recent high-profile thefts or other losses of cryptocurrency tokens worth huge sums of money. These include the collapse of Mt.Gox ($460 million stolen [5]), the Coincheck hack ($400 million stolen [6]), the DAO hack ($50 million stolen [7]), and the Parity bug ($160 million frozen [8]). The first two of these incidents involve 'traditional' OPSEC — trust, key, and information management issues that are no different for a cryptocurrency exchange than for a bank. However, the latter two incidents are of a different nature as the flaws themselves are *on the blockchain*. On one hand this is good for transparency, because the exact nature of the bug is publicly visible and the movement of stolen funds can be tracked. However, because blockchains are append-only, it is impossible to fix security flaws or recover lost funds without a hard fork. A similar point can be made for unintended consequences of the crypto-economic incentives that are inherent to blockchain consensus protocols. Examples include the emergence of mining pools, the potential of 51% attacks, or block withholding attacks against the entire network [9] or the incentive structure within pools.[1] Since solutions require profound changes to the protocol, hard forks again tend to become a necessity.

In this paper, we present an overview of the security challenges that are unique to blockchain technology, and argue that these are best confronted through extensive testing before deployment. The structure of the paper is as follows. In Section II we present a database of blockchain incidents that was compiled in a structured manner. From this database, we learn that while traditional OPSEC-related issues remain the largest source of incidents, a considerable proportion (roughly one third) of the incidents are blockchain-specific. These can further be subdivided into two categories: smart contracts and protocol incentives. These are discussed in Section III and Section IV respectively. Section V concludes the paper.

II. BLOCKCHAIN INCIDENT DATABASE

As mentioned in the introduction, the increased market capitalisation of cryptocurrencies, combined with the lack of maturity of the surrounding security ecosystem, has led to a considerable number of security incidents. Blockchain incidents are defined as attacks resulting in a loss of assets. These incidents have been reported in different sources and with different levels of detail.

In this section, we present a first structured attempt to collect and curate a database of blockchain incidents. The resulting database supports analytics that we believe will improve the cybersecurity of blockchain applications. With the blockchain incident database, the different risks can be identified that should at least be known to the blockchain community. The incident database can serve as a reference for blockchain developers to improve the security of their solutions, and it will also improve the understanding of known vulnerabilities

---

[1]https://www.reddit.com/r/Bitcoin/comments/28242v/eligius_falls_victim_to_blocksolution_withholding/

in blockchain deployments. Incident types span from traditional cyber attacks on cryptocurrency exchanges such as Mt. Gox [5], to exploits of vulnerable smart contracts [7] and hacked computers that mine cryptocurrencies [10].

A. Database Creation

Our goal is to create an easy-to-use database containing a representative set of blockchain incidents. The collection of the incidents consists of four steps: identification, description, classification, and review of incidents.

Identification: We assume that all relevant incidents can be identified in open sources [11]. First, there are specific sources, such as the "Blockchain Graveyard"[2] and the Ethereum Blog Security Archives. Additionally, Google searches for specific terms, such as "hacking for Bitcoin mining" and "conducted 51% attacks" provide further data. These sample queries are inspired by the security challenges identified by ENISA [12]. Given that this is a first attempt to build the database, we do not claim completeness. However, our ultimate goal is to develop a representative dataset through crowdsourcing.

Description: For each incident we store a description with ample references, such as screenshots, and web archives of original data for further verification. For the incident descriptions, we use STIX [13], which is an international standard for sharing cybersecurity incident data. We use 14 standard STIX objects as our fields, extended with a new field — 'Loss Estimation (crypto)' — describing the cryptocurrency involved, such as Bitcoin (BTC), Ether (ETH) or any altcoin lost in the incident.

The incidents can be viewed and edited using a database platform, which is running in the cloud and provided as a web service. Moderated registration on the platform will soon open.

Incidents are described with as many references as possible to give a view of the incident that is as broad as possible. Nevertheless, open sources are limited and we cannot guarantee the integrity of the complete picture. We aim to ensure verifiability of incidents by third parties. However, as most companies do not fully disclose information, we acknowledge that some information might be missing. Therefore, each incident is editable in order to add the latest insights.

Classification: Based on analysis of the first version of the incident database with 86 incidents, we propose three main classes of incidents:

OPSEC: Incidents compromising an organisation or individual's control of information and access to business-critical assets.

Smart Contracts: Incidents resulting from improperly written smart contracts deployed and executed on a blockchain.

Consensus Protocol Incentives: Incidents arising from malicious exploitation of consensus protocols that create opportunities and benefits for blockchain participants.

OPSEC is a container class of blockchain incidents based on traditional cyber attacks [14]. Smart Contract breaches are

[2]https://magoo.github.io/Blockchain-Graveyard/

further described in Section III, whereas Consensus Protocol Incentive vulnerabilities are discussed in Section IV.

Review: The final step in the process of building the database is the review, which ensures that independent parties verify incident data. Thus far we have employed students to provide a somewhat independent view on the incidents. Ultimately, we hope that we will be able to incentivise the blockchain community to contribute to the review process.

B. Incident Analytics

Most of the added value of the incident database is provided by the analytics, such as the time to incident discovery and the monetary losses incurred. For example, even for our relatively small collection of incidents covering the period 2011-2018, we estimate that some US$ 3.55B has been lost.

We propose to build a number of more advanced analytic capabilities as follows:

- Develop a deep understanding of the attack vectors within the different classes of incidents.
- Analyse the time from attack to detection, to see how cybersecurity changes as a result of blockchain technology evolution.
- Investigate the correlation between specific blockchain technologies and classes of incidents.
- Create a timeline of the losses based on incidents.
- Discover trends in the number of incidents and losses over time, to see whether countermeasures are effective.
- Map geographical origins of attacks.

Based on our first collection of blockchain incidents we can draw the following preliminary conclusions.

OPSEC: The majority of incidents in our current database occurred due to a lack of sufficient OPSEC measures (about 66%). The key blockchain component is the opportunity for attackers to confiscate enormous amounts of cryptocurrencies, resulting in incidents with millions of lost assets. To prevent the OPSEC class of incidents, standard cybersecurity solutions are available. For this reason, our position is to consider OPSEC out-of-scope regarding research to new solutions.

Smart Contracts: This category makes up about 22% of our incident database. These incidents occur when a smart contract does not work the way it was intended. The transparency of blockchains makes it possible to audit all published smart contracts, which can be supported by tools. For this reason, our position is to conduct research into new solutions focused on smart contract testing, as we discuss in Section III.

Consensus Protocol Incentives: Consensus protocol attacks are more difficult to detect than smart contract related attacks, as the effect is usually the improper mining of a block or the censorship of nodes. It is hard for miners to mitigate or even detect attacks. Furthermore, the majority of the incidents arising from incentives are due to second-order, unintended effects of blockchain use. These incidents make up roughly 12% of our database. Our position is to use a formal framework — see Section IV — to evaluate the incentives of

the consensus protocols in advance, instead of realising the criminal opportunities after an attack.

III. Smart Contract Security

The concept of smart contracts is a relatively new paradigm that enables different mutually untrusting parties to encode and enforce their agreements. Practical implementations of smart contracts emerged with the advent of blockchain platforms based on distributed consensus protocols. Currently, mainstream smart contract platforms, like Ethereum [15], follow the model of replicated execution. In this model, the code of all smart contracts, and all calls to contract methods, are irreversibly appended to a blockchain. Every node participating in the blockchain protocol reads code and calls from the blockchain, instantiates the same code, and executes the same calls, getting the same results and maintaining the same state.

From the security point of view, this model has many important implications. First of all, the development life cycle of smart contracts is significantly different from the traditional software development life cycle, where testing, integration, and maintenance are repeatable. Since a smart contract's code is unchangeable after being appended to a blockchain, developers have to bind code to a variable if they wish to modify the behaviour of their contracts later on. In that context, the development life cycle of smart contracts is much different from standard software that can be patched and fixed on-the-fly. One could even argue that as code changes are impossible, the development of smart contract is not a cycle anymore.

Subsequently, mistakes in code (like logical errors or even typos) or mistakes in the use of smart contracts (like calling a method with wrong parameters) are irreversible and costly. This argument is supported by our analysis of a sample contract as shown in Case Study I. In fact, hackers are constantly looking for vulnerabilities in deployed smart contracts, treating them as specific bug-bounty programs [16], where they can *get paid* for exploiting vulnerabilities.

Unfortunately, it is difficult to prevent such attacks as deployed code cannot be patched retrospectively. Instead, when a vulnerability is detected, a new smart contract has to be deployed to fix it. Although it is difficult for developers, the immutability of smart contracts is seen by the community as something positive. Consider, for example, a case where a developer could fix a bug in an already deployed smart contract. Such a change could cause collateral damage as other smart contracts might rely on the *fixed* functionality of the changed contract. Furthermore, what the developer of the smart contract considers a bug could be a property for developers of relying smart contracts. In that sense, smart contracts are similar to legislation (just to recall the sentence "Code is Law" [17]), and therefore should be stable and carefully prepared before deployment.

Our observations lead us to the conclusion that verification and testing are especially important in smart contract development, and should be an integral part of the analysis and design steps (and not only follow the implementation step as in the traditional development life cycle).

A. Testing and Verification

The aim of security testing is to determine whether a program has a vulnerability that can be exploited by an attacker [18]. Smart contracts are programs and may therefore contain vulnerabilities [19]. Smart contracts are usually short but they are also concurrent [20] and permanent, so probably more difficult to get right than ordinary programs [21].

There is a relatively recent review of smart contract vulnerabilities [22], which shows that smart contracts have classical issues (like off-by-one errors) as well as specific issues (like running out of gas). Smart contracts should therefore be subjected to security analysis and testing.

There are several types of testing and analysis tools that can be used on smart contracts, such as running hand crafted tests on a fast test network with the truffle framework[3], fuzzing of the input of the contract, mutating of the code of the contract [23], static analysis of properties of the contract [24], model checking of behaviours of a model of the contract [25], and theorem proving of properties of the program [26]. There are also runtime verification techniques, such as proof carrying code [27].

Some tools require test engineers to be highly skilled. For example the use of theorem provers requires considerable skill, whereas automatic static analysis tools[4] are simple to use. Ease of use does not necessarily mean that the results are uninteresting; this depends on the sophistication of the tool.

We believe that there is enough low-hanging fruit for us to be able to afford focussing on tools that are easy to use, such as mutation and fuzzing tools. We are unaware of existing literature on the application of fuzzing and mutating to smart contracts. These techniques can form the basis of a powerful test suite — i.e., one that has good code coverage and helps the test engineer to distinguish successful traces from failures.

Writing tests is both tedious and error prone. Fuzzing varies the inputs to a program under test, which helps to extend the test suite. Mutating varies the code of a program under test, which helps to increase the test coverage. However, even with these tools significant creativity is needed on the part of the test engineer to develop a comprehensive test suite for a program or smart contract.

We list four approaches that we believe can help the test engineer. These are all low-risk high-return approaches, building on a wealth of related work. Most approaches regard a contract from a different point of view, which gives the test engineer insights that he/she might not otherwise have acquired. For example, if a lottery contract has never paid out, a security problem is likely.

1) Documentation for developers and testers: The first approach is to improve the documentation on smart contract development and testing. It is difficult to develop smart contracts because the documentation is scant and poorly organised. The Ethereum yellow paper [28] can be hard to read, the official documentation of the Solidity language is primarily

---

[3]See http://truffleframework.com
[4]See for example https://securify.ch

a collection of examples [29], there are many resources that tell parts of the story and invariably the developer ends up searching the web[5] for questions that might be related to the issues, hoping to find usable answers.

Consider as an example how to determine whether a transaction has run out of gas. The Byzantium version of Ethereum returns a status flag in the transaction receipt to indicate this. Other versions require the programmer to compare the actual amount of gas used to the maximum amount of gas passed to the transaction. If the two are equal, the probability is high that indeed the transaction has run out of gas. The probably is not 100%, since it is theoretically possible that the transaction used exactly the maximum amount of gas. If the developer knows what he/she is looking for, the information can be found. But many unsuspecting developers will probably make mistakes because they cannot find the information they are looking for.

We believe that a handbook for smart contract developers and testers is long overdue.

2) Fuzz the inputs to smart contracts: The second approach is to fuzz the inputs of the smart contracts. Fuzzing has been proved as an effective and fast technique for finding security vulnerabilities [30], [31]. It can be applied directly to a smart contract by applying invalid and unexpected inputs and observing behaviour of the smart contract. Fuzzing can also be used for finding bugs specific to smart contracts. For example, by varying the limit on the available amount of gas, transactions can be aborted at arbitrary points, possibly leaving the contract in an insecure state. Another example is to randomise the order of calls, as in some cases a miner or underlying network can influence this, and observe the impact.

3) Mutate the code of smart contracts: The third approach is to develop a mutation tool for smart contracts. This has been proposed before,[6] but the attempt has been abandoned. The challenge is for the mutation generator to understand enough of the semantics of the smart contract language to generate only useful mutants. One possible route is to export

[5]See for example https://stackoverflow.com

[6]See https://github.com/ethereum/solidity/issues/1172

Case Study II: Solidity compiler bugs introduce vulnerabilities in high-value contracts

We analysed the top 1,000 contracts by balance of the almost 20,000 verified smart contracts on etherscan.io. A contract is verified if its source code and compiler version are made public.[d] Making this information public improves transparency and should therefore increase the confidence in the contract.

The contract with the largest balance holds 1,500,000 ETH, the smallest balance is just 0.9 ETH. The sum total of the balance of the 1,000 contracts is just over 5 million ETH, or about 5% of the total amount of ETH tokens in circulation. Therefore we believe our selection of contracts to be representative for

n	version	release date
120	v0.1-3.*	2015
88	v0.4.0-9	2016
14	v0.4.10	Mar 2017
68	v0.4.11	May 2017
41	v0.4.12-14	Jul 2017
122	v0.4.15-16	Aug 2017
36	v0.4.17	Sep 2017
187	v0.4.18	Oct 2017
207	v0.4.19	Nov 2017
78	v0.4.20	Feb 2018
39	v0.4.21	Mar 2018
1000	total	

contracts on Ethereum.

Some of the 1,000 verified contracts were compiled by a version of the Solidity compiler that is more than a year old. Older versions have known bugs that introduce vulnerabilities in the compiled code. The table to the left lists how many contracts (n) have been compiled with the various compiler versions.

Of the 1,000 verified high-value contracts, almost half has a vulnerability caused by a known bug in the Solidity compiler. The following table lists how many contracts have a low/medium/high severity issue:

About one third of the verified high-value contracts has a medium to high severity vulnerability. Since a contract is baked into the blockchain, it cannot simply be recompiled with an up-to-date version of the compiler to remove the vulnerabilities. This is clearly an issue of smart contracts that should be resolved. Again this case study supports our position that more testing is needed.

#contracts	severity level
128	high
192	medium
10	low
158	very low
488	total

[d]See https://etherscan.io

the abstract syntax tree from the smart contract compiler, mutate it, and import the mutant again in the compiler.

4) Search the blockchain for tests: The fourth approach to address the problem of insufficient tests is to search the blockchain for traces of already deployed smart contracts. The blockchain contains all code and data needed to recreate the state of the world at each time a block was mined. Therefore, the traces from the blockchain contain enough information to generate tests. This idea is similar to using the counter examples produced by model checkers for testing purposes [32]. The difference is that we do not have to create a model of the contract; instead the trace originates from the contract itself. Both testnet and livenet deployments of smart contracts can be searched for traces. These deployments are probably not fully tested; hence the test engineer will have to inspect the generated tests carefully. We believe that such generated tests could complement a test suite written by hand.

Most of the techniques that we describe in this chapter exist already. However, the Dapps that we have analysed show that not all developers master the relevant techniques. So there is a need to make these more accessible, starting by improved documentation.

B. Secure Smart Contracts Distribution

Since smart contract platforms have associated cryptocurrencies, they are attractive medium for fraud, often imple-

mented as smart contracts. In traditional software distribution there are many models to handle such problems. For instance, open-source projects are driven by communities who have interest in fixing bugs and increasing the quality of these projects. Similarly, centralised software repositories (like Apple App Store, Google Play, or Ubuntu repositories) are controlled by companies who have incentives in removing malicious code, as it directly affects their clients.

By contrast, the distribution of smart contracts does not follow any of these models. Smart contracts are submitted by developers (anyone can act as a developer) and are appended to a blockchain by miners (i.e., nodes that participate in the underlying consensus protocol). The irreversibility and censorship-resistance of smart contract platforms do not allow anyone to remove an added smart contract (even if it evident that it is malicious), so providing security in such an architecture is a challenging task.

In order to improve on this, we propose a proactive smart contract publishing architecture. The main observation is that miners of smart contract platforms have conflicting incentives:

- they have incentive to add every smart contract submitted (even malicious), as appending code to the blockchain rewards them, and
- they have incentive to keep the ecosystem secure, as otherwise it might lose popularity affecting their rewards.

A high-level idea of our architecture is to keep adding all

submitted smart contracts to the blockchain, and to keep testing their security properties. Then for every found (potential) vulnerability a warning can be signalled in the blockchain, such that everyone can see it. In particular we distinguish two deployment models of such an architecture.

In the first model, a miner before adding a smart contract tests it with various testing methodologies and tools (see Section III-A). The miner publishes the smart contract along with the obtained testing results. For instance, these results can indicate that the smart contract has a potential backdoor or a kill switch, or it has some features of a Ponzi scheme [33]. When appended to the blockchain these results are immutable and visible to everyone, thus can act as reliable warnings. An alternative deployment model, is to introduce a dedicated service which would constantly test existing and incoming smart contracts, and would publish testing outcomes (i.e., warning) at a pre-defined location. An advantage of this approach, is that such a service can be upgraded to include new test methods and vectors, and does not depend on miners who might be be unwilling to conduct additional work [34]. We leave details of this architecture as a future work, however, we envision that such a service can be maintained by community contributors and its security can be enhanced by means like a trusted execution environment.

IV. CRYPTOECONOMIC PROTOCOLS & THE PRESTO FRAMEWORK

At the core of blockchain platforms such as Bitcoin and Ethereum lies the emergence of new protocols which aim to align the interests of several self-interested parties that do not necessarily trust each other. Different platforms combine different ideas and explore different trade-offs. These ideas can be better understood as points in a high-dimensional space, and the goal of a protocol designer is to develop protocols that lie on the Pareto frontier of the design space, i.e., systems whose desirable properties cannot all be simultaneously improved.

An important tool towards understanding the design space better is the organisation of "desirable" properties onto different axes. This organisation allows to perform a type of divide-and-conquer approach where we can at a first step explore the fundamental limitations of what is achievable within each category (or different combinations of these categories) as well as to explore connections with other well established disciplines in computer science and mathematics in general.

Here we briefly present the basics of the PRESTO framework [35], which is an acronym for Persistence, Robustness, Efficiency, STability, and Optimality (see Fig. 1). We discuss each of these below, in reverse order:

Optimality: Optimality concerns the question of whether the protocol maximises the quality of certain outcomes. Ideally the protocol should be mathematically proven to provide these guarantees.

Optimality is one of the most basic properties that a protocol can satisfy. It can be ascertained using mathematical analysis techniques that range from the very basic (e.g., calculus) to the more advanced (e.g., optimisation theory) [36], [37]. In
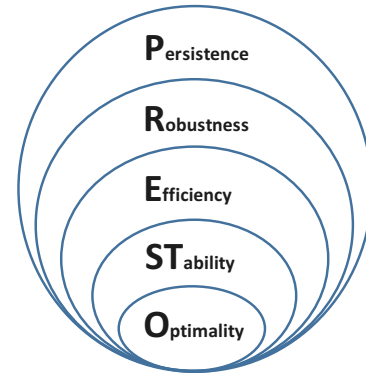

Fig. 1. Graphical representation of the PRESTO framework.

the context of blockchain protocols, examples of optimality include formal proofs of liveness and safety under "standard" network/agent conditions — see, e.g., [38] for an overview of these properties for several recently implemented consensus protocols that are based on Byzantine fault tolerance.

Stability: Stability means that it is in the participants' best interest to follow the rules of the protocol. In concrete terms, this means that the protocol is a Nash equilibrium (or more generally an incentive compatible outcome, e.g. a dominant strategy). An equilibrium is an outcome that is optimal from the perspective of all decision makers involved.

This has been reasonably thoroughly studied (since 1940s, Nash [39], von Neumann [40]). Game theory and economics provides numerous tools to model and analyze such settings. Importantly, stability does not imply optimality. The "price of anarchy" literature studies exactly this tension between equilibration and efficiency of outcomes [41] and shows how in many natural settings equilibria although not perfectly optimal can be approximately optimal. Mechanism design on the other hand aims to re-design the rules of games so that individual incentives are perfectly aligned with societal goals. For blockchain protocols, stability means that it is best for agents in the default setting to follow the reference protocol.

Efficiency: Does the protocol make efficient use of its computational resources? Does it perform its core tasks as fast as possible, using as little space as possible, using as little randomness as possible, using as little energy as possible, and using parallelisation efficiently?

Efficiency of computation, at least from the perspective of time and space, has been reasonably thoroughly studied (since 1940s, Turing, von Neumann). Computational complexity theory provides the most carefully designed framework for studying the fundamental limitations of efficient computation [42]. These limitations force us to consider trade-offs, e.g., approximate optimality versus speed (e.g., approximation algorithms [43]). Ideally, a protocol implements a (near) optimal equilibrium efficiently. Algorithmic game theory and mechanism design study questions on the intersection of optimality, efficiency and stability [44]. For a blockchain example, the use of computational resources by the Bitcoin protocol is not

efficient: the maximum transaction throughput is the same as five years ago despite a dramatic increase in hash rate and energy consumption [45].

Robustness: Suppose that the protocol is reasonably close to an optimal efficient equilibrium on paper. Real distributed systems pose more challenges (e.g. asynchrony, communication delays, users might have different utility functions due to differences in electricity/computation costs/risk attitudes, collusion, etc.).

This is an emerging subfield within algorithmic game theory. A particular question of interest is what happens in the case of coalition formation or collusion. E.g., in [9] it is shown that if $x > \frac{1}{3}$ fraction of the total mining power is owned by a mining pool, then the following the Bitcoin protocol is not an equilibrium. Specifically, the paper describes a strategy that can be used by a minority pool to obtain more revenue than the pool's fair share, i.e., more than its ratio of the total mining power. In [46], [47] more elaborate attacking strategies are explored, as well as nearly-tight thresholds on the computational power of the attacker under which the honest strategy remains a Nash equilibrium.

Persistence: Finally, what if the protocol is subjected to a severe attack or black-swan event, can the system recover? How fast, and at what cost? For persistence, we take this idea to its logical extreme. We assume that the system may be always under attack, and design it so that it recovers and provides the desirable properties often.

Formally, we define a *strongly persistent* property to be eventually be satisfied (and stay satisfied) given any initial system condition. By contrast, a *weakly persistent* property will eventually be satisfied given any initial system condition and will become satisfied again infinitely often [48]. Such ideas have been introduced within evolutionary game theory [49] as well as the study of biological systems (i.e. recovery of a ecosystem after infection from a virus) [50], however, the combination of these ideas with tools from optimisation theory and algorithm design have not been really explored. [48], [51] perform some exploratory steps in this direction.

A desirable property is not satisfied by a system just in equilibrium, but it is satisfied in a dynamic way. This allows for more flexibility to explore trade-offs between recovery/convergence time, 'periodicity', and the cost of implementation. Note that two (or more) incompatible properties can both be supported in a weakly persistent manner.

Summing up, the PRESTO framework sees protocols as a nesting doll with the following cascade of goals. First, optimality requires that the protocol solves the problem that it is defined to address, otherwise there is no good reason to deploy it and the designer should go back to the drawing board. Second, efficiency requires that resources are used as efficiently as possible (e.g. time, space, network bandwidth, energy, randomness, etc.). Next, stability analysis aims to make sure that self-interested agents have an incentive to follow and implement the protocol, i.e., that the protocol itself is an equilibrium. If not the agents will deviate from it and the deployed protocol will behave unpredictably in practice. Given an optimal, stable and efficient protocol we can then start considering more complicated behavioral models from the perspective of the agents. How robust is the equilibrium and its properties if we "perturb" the underlying assumptions, e.g., what if agents are risk-averse, or can form coalitions, how do the equilibria of these more realistic models compare to those idealised equilibria in the more vanilla settings? Finally, persistence goes beyond equilibrium thinking and asks which properties can be guaranteed not at equilibrium (i.e., consistently and at every single point in time), but maybe only recurrently (e.g., at consistent intervals, like once a week). This flexibility might allow us to provide guarantees that are impossible to satisfy in equilibrium. For example, two mutually exclusive properties cannot be both implemented at equilibrium, but can be guaranteed in a persistent manner by a system that cycles through different states.

Exploring these trade-offs is a fascinating area for multidisciplinary research that could help us improve upon the understanding and design of the current protocols, as well help incorporate ideas from other well-established fields (e.g., algorithmic game theory, evolutionary game theory, complexity theory, approximation algorithms, a.o.).

V. CONCLUSIONS AND RECOMMENDATIONS

In this paper, we have presented an overview of the security challenges in blockchain technology, and proposed ways to go forward. Our first step was to create the, to our knowledge, first structured attempt at cataloguing the wide range of blockchain-related security incidents. We find that apart from traditional OPSEC, many incidents are of a nature that is specific to blockchain technology. Two categories stand out: smart contracts, and techno-economic protocol incentives. In both settings, we argue that it is vital to properly test before deployment, because faults cannot be corrected without a hard fork. For the smart contracts, we have presented four testing approaches, building on the latest developments in automated code testing such as fuzzing, mutating, and model checking. For the protocol-level incentives, we have proposed a framework for comparing the security properties of protocols by ranking them across five dimensions. It is our position that for blockchain security to become mature, a rethink that leads to comprehensive testing before deployment is a necessity.

REFERENCES

[1] Marco Iansiti and Karim R Lakhani. The truth about blockchain. *Harvard Business Review*, 95(1):118–127, 2017.

[2] Sarah Underwood. Blockchain beyond Bitcoin. *Communications of the ACM*, 59(11):15–17, 2016.

[3] Melanie Swan. *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc., 2015.

[4] Don Tapscott and Alex Tapscott. *Blockchain revolution: how the technology behind Bitcoin is changing money, business, and the world*. Penguin, 2016.

[5] Robert McMillan. The inside story of Mt.Gox, Bitcoin's $460 million disaster, 2014. [Online]. Available: https://www.wired.com/2014/03/bitcoin-exchange/. [Accessed: 13-4-2018].

[6] Bryan Menegus. $400 million goes missing from Japanese crypto exchange Coincheck, 2018. [Online]. Available: https://gizmodo.com/400-million-goes-missing-from-japanese-crypto-exchange-1822454084. [Accessed: 13-4-2018].

[7] David Siegel. Understanding the DAO attack, 2016. [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists/. [Accessed: 13-4-2018].

[8] Rachel Rose O'Leary. Parity team publishes postmortem on $160 million Ether freeze, 2017. [Online]. Available: https://www.coindesk.com/parity-team-publishes-postmortem-160-million-ether-freeze/. [Accessed: 13-4-2018].

[9] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[10] Aaron Mak. How scammers steal your computing power to mine cryptocurrencies, 2018. [Online]. Available: https://slate.com/technology/2018/02/what-is-cryptojacking-the-bitcoin-and-monero-mining-process-that-steals-your-computing-power-explained.html. [Accessed: 13-4-2018].

[11] Michel van Eeten, Albert Nieuwenhuijs, Eric Luiijf, Marieke Klaver, and edite Cruz. The state and the threat of cascading failure across critical infrastructures: the implications of empirical evidence from media incident reports. *Public Administration*, 89(2):381–400, Jun 2011.

[12] European Union Agency for Network and Information Security (ENISA). Distributed ledger technology & cybersecurity: Improving information security in the financial sector. 2017.

[13] Sean Barnum. Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX). *MITRE Corporation*, 11:1–22, 2012.

[14] Kiran Nagaraj, LaDarius Goens, Sam Wyner, and Eamonn Maguire. Securing the chain, 2017. [Online]. Available: https://assets.kpmg.com/content/dam/kpmg/xx/pdf/2017/05/securing-the-chain.pdf. [Accessed: 13-4-2018].

[15] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform, 2014. [Online]. Available: https://whitepaperdatabase.com/ethereum-eth-whitepaper/. [Accessed: 13-4-2018].

[16] Andreas Kuehn and Milton Mueller. Analyzing bug bounty programs: An institutional perspective on the economics of software vulnerabilities. In *The 42nd Research Conference on Communication, Information and Internet Policy*, 2014.

[17] Lawrence Lessig. Code is law. *The Industry Standard*, 18, 1999.

[18] Bruce Potter and Gary McGraw. Software security testing. *IEEE Security & Privacy*, 2(5):81–85, Sep 2004.

[19] Maarten Everts and Frank Muller. Will that smart contract really do what you expect it to do?, Jan 2018. [Online]. Available: http://publications.tno.nl/publication/34626215/rNvlPU/TNO-2018-smart.pdf. [Accessed: 13-4-2018].

[20] Ilya Sergey and Aquinas Hobor. A concurrent perspective on smart contracts. In *Int. Conf. on Financial Cryptography and Data Security (FC)*, pages 478–493, Sliema, Malta, Apr 2017. Springer.

[21] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *Financial Cryptography and Data Security (FC)*, volume 9604 of *LNCS*, pages 79–94, Barbados, Feb 2016. Springer.

[22] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on Ethereum smart contracts (SoK). In *6th Conf. on Principles of Security and Trust (POST)*, volume 10204 of *LNCS*, pages 164–186, Uppsala, Sweden, Apr 2017. Springer.

[23] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Trans. Softw. Eng.*, 37(5):649–678, Sep 2011.

[24] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pages 254–269, Vienna, Austria, Oct 2016. ACM.

[25] Kaylash Chaudhary, Ansgar Fehnker, Jaco van de Pol, and Marielle Stoelinga. Modeling and verification of the Bitcoin protocol. In *1st workshop on Models for Formal Analysis of Real Systems (MARS)*, volume EPTCS 196, pages 46–60, Suva, Fiji, Nov 2015.

[26] Yoichi Hira. Defining the Ethereum virtual machine for interactive theorem provers. In *Int. Conf. on Financial Cryptography and Data Security (FC)*, volume 10323 of *LNCS*, pages 520–535, Sliema, Malta, Apr 2017. Springer.

[27] Sönke Holthusen, Michael Nieke, Thomas Thüm, and Ina Schaefer. Proof-carrying apps: Contract-based deployment-time verification. In *Int. Symp. on Leveraging Applications of Formal Methods (ISoLA)*, volume 9952 of *LNCS*, pages 839–855, Corfu, Greece, Oct 2016. Springer.

[28] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. Technical report 759dccd, Ethcore.io, Aug 2017. [Online]. Available: hhttps://ethereum.github.io/yellowpaper/paper.pdf. [Accessed: 13-4-2018].

[29] Ethereum. Solidity documentation, Mar 2018. [Online]. Available: http://solidity.readthedocs.io/en/v0.4.21/. [Accessed: 13-4-2018].

[30] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[31] Patrice Godefroid, Michael Y Levin, and David Molnar. Sage: whitebox fuzzing for security testing. *Communications of the ACM*, 55(3):40–44, 2012.

[32] Gordon Fraser, Franz Wotawa, and Paul E. Ammann. Testing with model checkers: a survey. *Software Testing Verification and Reliability*, 19(3):215–261, Sep 2009.

[33] Massimo Bartoletti, Salvatore Carta, Tiziana Cimoli, and Roberto Saia. Dissecting ponzi schemes on ethereum: identification, analysis, and impact. *arXiv preprint arXiv:1703.03779*, 2017.

[34] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 706–719. ACM, 2015.

[35] Georgios Piliouras and Vlad Zamfir. Personal Communication, 2017.

[36] Philip E Gill, Walter Murray, and Margaret H Wright. *Practical optimization*. Academic press, 1981.

[37] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[38] Christian Cachin and Marko Vukolić. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[39] J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, pages 48–49, 1950.

[40] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[41] Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 404–413, 1999.

[42] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[43] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[44] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[45] Karl J O'Dwyer and David Malone. Bitcoin mining and its energy footprint. In *Proceedings of the 25th IET Irish Signals & Systems Conference*, pages 280–285, 2014.

[46] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532. Springer, 2016.

[47] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 365–382. ACM, 2016.

[48] Georgios Piliouras, Carlos Nieto-Granda, Henrik I Christensen, and Jeff S Shamma. Persistent patterns: Multi-agent learning beyond equilibrium and utility. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 181–188. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[49] J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, Cambridge, 1998.

[50] Hal L Smith and Horst R Thieme. *Dynamical systems and population persistence*, volume 118. American Mathematical Soc., 2011.

[51] Georgios Piliouras and Jeff S Shamma. Optimization despite chaos: Convex relaxations to complex limit sets via Poincaré recurrence. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 861–873. SIAM, 2014.