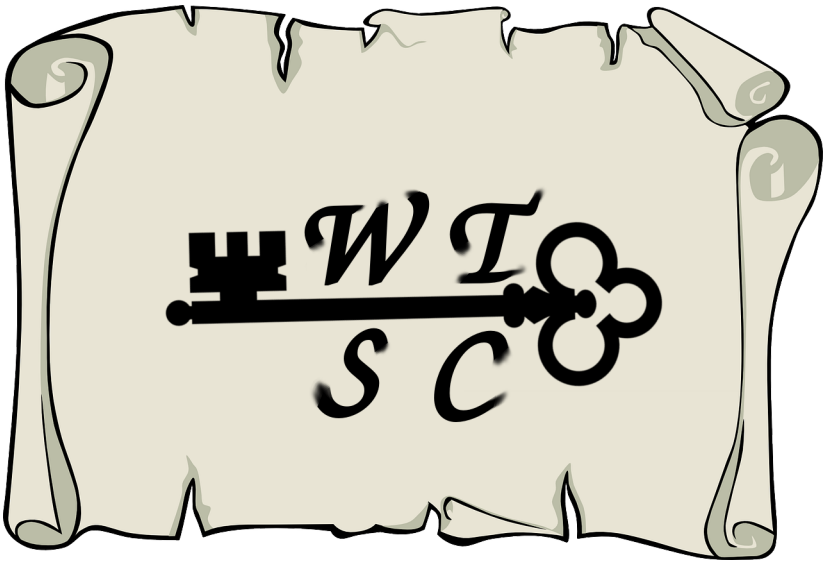


SECOND WORKSHOP ON TRUSTED SMART CONTRACTS 2018 (WTSC18)

Curaçao, 2nd March 2018

http://fc18.ifca.ai/wtsc/

WTSC18 Program Committee

Marcella Atzori	UCL, UK / IFIN, IT	
Daniel Augot	INRIA, FR	
Massimo Bartoletti	University of Cagliari, IT	
Devraj Basu	Strathclyde University, UK	
Stefano Bistarelli	University of Perugia, IT	
Alex Biryukov	University of Luxembourg, LU	
Andrea Bracciali	University of Stirling, UK	(chair)
Daniel Broby	Strathclyde University, UK	
Bill Buchanan	Napier University, UK	
Martin Chapman	King's College London, UK	
Tiziana Cimoli	University of Cagliari, IT	
Nicola Dimitri	University of Siena, IT	
Stuart Fraser	Wallet.services, UK	
Neil Ghani	Strathclyde, UK	
Davide Grossi	Utrecht University, NL	
Oliver Giudice	Banca d'Italia, IT	
Yoichi Hirai	Ethereum DEV UG, DE	
Ioannis Kounelis	Joint Research Centre, European Commission	
Victoria Lemieux	The University of British Columbia, CA	
Loi Luu	National University of Singapore, SG	
Carsten Maple	Warwick University, UK	
Michele Marchesi	University of Cagliari, IT	
Fabio Martinelli	IIT-CNR, IT	
Peter McBurney	King's College London, UK	
Neil McLaren	Avaloq, UK	
Philippe Meyer	Avaloq, UK	
Bud Mishra	NYU, USA	
Carlos Molina-Jimenez	University of Cambridge, UK	
Federico Pintore	University of Trento, IT	(chair)
Massimiliano Sala	University of Trento, IT	(chair)
Ilya Sergey	UCL, UK	

Thomas Sibut-Pinote	INRIA, FR
Jason Teutsch	TrueBit Establishment, LIE
Roberto Tonelli	University of Cagliari, IT
Luca Vigano'	University of Verona, IT
Philip Wadler	University of Edinburgh, UK
Santiago Zanella-Beguelin	Microsoft, UK

WTSC18 Overview

These proceedings collect the papers accepted at the *Second Workshop on Trusted Smart Contracts (WTSC18)* associated to the Financial Cryptography and Data Security 2018 (FC18) conference held in Curaçao in 2018 (February 26–March 2, 2018).

WTSC18 focused on *smart contracts*, i.e. self-enforcing agreements in the form of executable programs, and other *decentralised applications* that are deployed to and run on top of (specialised) blockchains. These technologies introduce a novel programming framework and execution environment, which, together with the supporting blockchain technologies, carry unanswered and challenging research questions. Multidisciplinary and multifactorial aspects affect correctness, safety, privacy, authentication, efficiency, sustainability, resilience and trust in smart contracts and decentralised applications.

WTSC18 aimed to address the scientific foundations of Trusted Smart Contract engineering, i.e. the development of contracts that enjoy some verifiable "correctness" properties, and to discuss open problems, proposed solutions and the vision on future developments amongst a research community that is growing around these themes and brings together users, practitioners, industry, institutions and academia. This was reflected in the multidisciplinary Programme Committee of this second edition of WTSC, comprising members from companies, universities, and research institutions from several countries worldwide, who kindly accepted to support the event. The association to FC18 provided an ideal context for our workshop to be run in. WTSC18 was partially supported by the University of Stirling, UK, the University of Trento, IT, and FC18 IFCA-ICRA.

This second edition of WTSC18 received fourteen submissions by about thirty authors, of which eight were accepted after peer review as full papers, and have been collected in the present volume. These analysed the current state of the art and legal implications of smart contracts; addressed aspects of security and scalability; proposed protocols for sealed-bid auctions, for lending cryptocurrencies, for distribution and managements of digital certificates; introduced logging schemes and models theorem-proving-based verification for smart contracts.

WTSC18 also enjoyed Arthur Breitman (Tezos Founder) and Bud Mishra (NYU, USA) as keynote speakers. Arthur gave a talk on present and future perspectives on models for Smart Contracts, while Bud presented a model for decentralised drug development.

Andrea Bracciali
Federico Pintore
Massimiliano Sala

WTSC18 Organisers

Home
Call for Papers
Program
FC 2018

2nd Workshop on Trusted Smart Contracts

March 2nd 2018

Accepted papers and Programme

09:00-10:00 **Invited Talk I (courtesy of the Bitcoin Workshop)**

The Blockchain Consensus Layer and BFT
Dahlia Malkhi

WMWare Research

10:00-10:30 **Joint Session with Bitcoin Workshop**

10:00 Smart Contracts for Bribing Miners
Patrick McCorry, Alexander Hicks and Sarah Meiklejohn

10:30-10:50 **Coffee Break**

10:50 **Welcome** and presentation of the **Open Vote Network on Ethereum** live experiment

10:55-11:45 **Invited Talk II : Applications**

BURPA or BUST! How to build a Bio-Unified Research Program Agency?
Bud Mishra

Courant Institute of Mathematical Sciences and NYU School of Medicine
New York University NYU, and
Mt Sinai School of Medicine

11:45-12:40 **Session 2**

11:45 Ghazal: toward truly authoritative certificates for secure web browsing
Seyedehmahsa Moosavi and Jeremy Clark

12:03 Verifiable Sealed-Bid Auction on the Ethereum Blockchain
Hisham Galal and Amr Youssef

12:21 The Game among Bribers in a Smart Contract System
Lin Chen, Lei Xu, Zhimin Gao, Nolan Shah, Ton Chanh Le, Yang Lu and Weidong Shi

12:40-14:00 **Lunch - Location: Shore**

14:00-14:55 **Invited Talk III**

Models for Smart Contracts: present and future perspectives

Arthur Breitman

Tezos

14:55-15:35 **Session 3**

14:55 Lightweight Logging over the Blockchain for Data-Intensive Applications

Yuzhe Tang, Zihao Xing, Cheng Xu, Ju Chen and Jianliang Xu

15:15 Comparative Analysis of the Legal Concept of Title Rights in Real Estate and the Technology of Tokens: How Can Titles Become Tokens?

Oleksii Konashevych

15:35-16:00 **Coffee Break**

16:00-16:55 **Round table: Voting, governance and decentralised democracy on blockchain**

Panel (tbc)

Jeremy Clark (Concordia), Peter Ryan (Luxembourgh), Arthur Breitman (Tezos), Bingsheng Zhang (IOHK), Christopher Allen (Blockstream), Andrea Bracciali (Stirling)

Demo Open Vote Network on Ethereum and live experiment's results

Patrick McCorry

16:55-17:55 **Session 4**

16:55 Proof-Carrying Smart Contracts

Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, Eric Koskinen and Vikram Saraph

17:15 The Scalability of Trustless Trust

Dominik Harz and Magnus Boman

17:35 Toward Cryptocurrency Lending

Chidinma Okoye and Jeremy Clark

17:55 **Closing**

This conference is organized annually by the International Financial Cryptography Association in cooperation with IACR.

WTSC18 Papers

BURPA or Bust!
How to build a Bio-Unified Research Project Agency? (Extended Abstract)

Bud Mishra[1]
In collaboration with: Q. Qi, L. Rudolph, F. Savas, and M. Weill

Courant Institute, NY 10012, USA,
mishra@nyu.edu

Abstract. A specter of sky rocketing drug prices is haunting the global health care. A novel market micro-structure in alliance with theories from financial engineering, smart contracts, systems biology and information asymmetric games can exorcise this specter, thus enabling lower per-patient costs for both curative and non-curative therapies for acute and chronic diseases, respectively, while accelerating research on drug discoveries.

Here, we formalize our and others' earlier design of mega-funds via an information-asymmetric signaling game model and then implement it with verifiable smart contracts. The model not only elucidates how the stakeholders strategically interact in this market using deception, adverse selection, moral hazards, etc. but also how to tame their interactions to improve the overall performance. In particular, we suggest and rigorously evaluate an embodiment built on a scalable implementation of smart contracts and crypto-currencies. Using extensive simulations, we show that, in the smart-contract-based mega-fund both senior and junior tranche investors get their principals fully repaid in 99.9%of the time.

In costly signals, we trust; it's BURPA or bust!

Keywords: Smart Contracts, Information Asymmetric Games, Cancer Megafund

1 Problem with Current Bio-Research Project Models

The pharmaceutical industry faces a significant barrier against accelerating research for drug discovery: specifically, for cancer. The average cost of new drug development in the U.S. was around USD 2.6 billion in the past ten years – up from an average of USD 1 billion in the 1990s. Such extraordinarily high costs for drug development are not only reflected in skyrocketing prices of approved cancer drugs, which thus places a substantial burden on U.S. households, but also in discouraging the pharmaceutical industry from allocating research-and-development (R&D) resources to projects with narrow profit margins. This state of affairs leaves many cancer subtypes, rare genetic disorders or third-world infectious diseases, all but neglected. Subsequently, it can result in critical medical needs remaining largely unmet. According to many scholars, such high costs, and by inference, the declining efficiency of R&D investment in biomedical industry, is a major indicator that the current biomedical business model is flawed, relying too heavily on the incentive provided by patents to rationalize the risk of investing in biomedical R&D.

1.1 Four Stages of Bio-Research Projects

Specifically, there are four stages for each biotechnological research: (*i*) funding; (*ii*) innovation and research; (*iii*) clinical trials and regulation approval; (*iv*) pricing and marketing (See Figure 1). Based on these stages, we categorize all the current research processes into two major approaches:

1. Not-for-profit approach: Government agencies (or charity and other nongovernmental organizations) collect funds from tax payers (or general public), and then distribute them to researchers in the university labs or research institutions based on peer review process. If a researcher makes a breakthrough, he either starts up an enterprise through a technology transfer office or licenses it to pharmaceutical companies.
2. Market approach: Pharmaceutical companies collect funds from venture capitalists or the capital market and hire researchers to work on promising projects. If a drug is approved by FDA, companies price it and sell it in the market or to the hospitals.

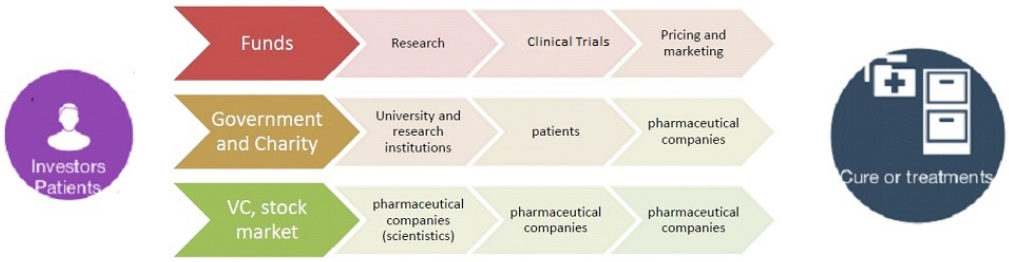


Fig. 1: *Stages of traditional biotechnology research.* The resulting market microstrucrures struggle with various forms of adverse selection, deception and moral hazards, which result from information asymmetry, exacerbated by mis-aligned utilities, lack of a community wide social norms and enforceable complete contracts.

In both of these approaches, participants from each stage usually have conflicts of interest, leading to fragmented strategies and agenda, which stifle information sharing across research teams necessary to advance treatments and cures.

Specifically, the not-for-profit approach suffers from the following three problems. First, government or non-profit organizations end up inefficiently allocating resources. For example, politicians may steer taxpayers money away from drug development to serve their own interests. As a result, under-investment is always an issue for not-for-profit drug development. Second, the interests of researchers and patients are not fully aligned. Purely academic competition among researchers, while beneficial in basic science research, prohibits them from sharing research results transparently and only a small percentage of NIH-funded medical research yields positive results that end up in publications. Third, the drug price is determined by pharmaceutical companies, therefore there is no control by the taxpayer or patients (who through charity invest their money in the first place) to rationally but strategically interact with the other key players.

1.2 Information Asymmetry and its Effects

Although the market approach is more efficient in allocating resources, pharmaceutical companies have traditionally taken advantage of its inherent information asymmetry to maximize corporate profit by charging high drug prices for the patients. In addition, they may only focus on disease groups that promise blockbuster returns and leave many rare diseases untreated. Powerful Big Pharmas also hold patents, trade secrets, know how, copyright, etc. for processes associated with actionable biomarkers and molecules. They are able to charge significant licensing fees to researchers who use or work on these intellectual assets, imposing barriers to information sharing within the drug development community. Thus, for instance, while personalized therapies are touted as the future of biomedicine, it is practically impossible to motivate a cohort to participate with their genomic data, as it will only deliver mostly equivocal, uninterpretable or non-actionable biomarkers, but not much else.

2 Combining with Crypto-currency and Smart Contracts

Given a biomedical research question (Example: "Can a drug [e.g., Avastin] be developed that will target hypoxic breast tumor cells by inhibiting angiogenesis?"), at least three types of resources are needed for the necessary research to gain momentum. These resources are: (*i*) research efforts from researchers (who understand angiogenesis pathways), (*ii*) capital provided by investors (who perform due diligence on suitability, safety and efficacy of Avastin ([Bevacizumab]) and (*iii*) data from patients attending clinical trials (who suffer from advanced breast cancer with VEGF mutation). Now, suppose a type of contract (possibly associated with a currency) existed and represented the ownership of all the pharmaceuticals, as well as the intellectual properties produced during this drug development process (henceforth, biomedical-research-based crypto-currency shortened, crypto-currency). This currency could then be earned by patients if they attend clinical trials, could be earned by researchers if they conduct experiments for this research question, and could be bought by investors. We may then conclude that the currency's versatility will lead to patients, researchers and investors from everywhere identifying themselves and noncoercively contributing to this drug development processes. In addition, if every participant in the drug development process owned part of the final products, their interests would be automatically aligned, as they receive nothing if the drug development process fails.

In order for this currency to serve as the norm, we need to make sure that all participants in the drug development believe this currency has value. The key difference between a drug development process and bitcoin mining is that drug development is highly experimental in the sense that its success rate depends highly on the talents, stamina and perseverance of researchers, and its outcome cannot be evaluated solely by computer models in silico.

2.1 Institutions

Therefore, we propose the following three (virtual) institutions to facilitate the drug discovery process and honor the commitment by the cryptocurrency.

1. **A cryptocurrency mega-fund** that sets predetermined rules for the open innovation research process, constructs diversified portfolios of these research projects, issues cryptocurrency to represent the ownership of these portfolios and honors the commitment of the cryptocurrencies. It is worth noting parenthetically that the organizational structure of this cryptocur-

rency mega-fund is fundamentally different from that of the mega-fund proposed by Fagnan et al.(2013)(henceforth, centralized mega-fund). In the centralized mega-fund, the fund managers need to optimize decisions of capital structure, through buying and selling compounds for each experiment, as well as by hiring and contracting with researchers in each stage of the drug development. The possibility of a misalignment of interests between fund managers and investors – one of the primary reasons behind 2007-2009 financial crisis – could significantly reduce the profitability of such a mega-fund. The crypto-market mega-fund would overcome this concern by using a decentralized, transparent, and market-based solution for drug development. All the activities during the open innovation research process follows predetermined rules. By avoiding a central authority governing the market and other transitional institutions, it avoids non-transparency and deception associated with the market manipulation. It also globalizes the system and encourages scaling with liquidity.

2. **A blockchain ledger** ensuring that all predetermined rules will be implemented as contracts with minimum costs. Specifically, the mega-fund manages each research project through smart contracts and real-time accounting. For investors, the costs of collecting accounting information and of enforcing the contracts are almost zero. For researchers, the funds for each stage of research will be distributed automatically if they meet predetermined milestones, their use of the cryptocurrency will be recorded in a real-time accounting system, and their discoveries will be time stamped on the blockchain. Now they have a cheaper and faster way to protect their intellectual property rights, alternative to patenting them. By compensating innovation with the cryptocurrency, researchers contributions will subsequently find their way into a commercial product, and they are then entitled to a statutory share of the products revenues.
3. **A secondary exchange market** of the cryptocurrency which would give liquidity to patients, investors and researchers. Encouraged by the cost-efficient feature of the cryptocurrency mega-fund, pharmaceutical companies would want to collaborate, instead of competing, with this mega-fund. They, along with health insurance companies, public health organizations (e.g. CDC) and charities could join the exchange market as market makers. Namely they could buy big blocks of these cryptocurrency based on their estimation of the demand, and subsequently sell it to future patients. New occupations would then emerge and will include data analysts, who will estimate returns and risks using translational systems biology and machine learning and then price the cryptocurrency to help investors better understand the research feasibility and progress.

3 The Importance of Smart Contracts

In our approach, more productive researchers will get much more research funds and higher compensation than current system. In fact, there are three design goals for the new funding system: (*i*) more innovative, efficient and productive researchers will get more funds; (*ii*) the researchers will be paid based on their performance, therefore the return to researchers with breakthrough discoveries under the new system will be significantly higher than those under the current system; (*iii*) the researchers are encouraged to take risks. In other words, they need not fear of being punished for failure when trying innovative approaches.

To design a funding system with these features, we need a better understanding of the following two questions: (*i*) *What is the production function of knowledge*? (*ii*) *What is the best way to motivate researchers*? These are questions that have been studied for decades, yet no consensus has

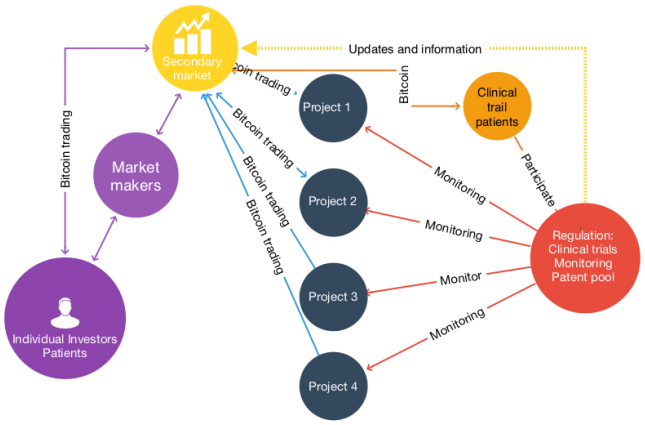


Fig. 2: *Crypto-currency system.* It tames the information asymmetry with flows of information balanced by reciprocating flows of obligations and rights (via investments and smart-contracts). Furthermore, it seamlessly includes all stake-holders: patients, researchers, investors and regulators. For example, patients invest on research based on disease risks, receive crypto-currencies and acquire the rights to buy drugs from research upon disease onset.

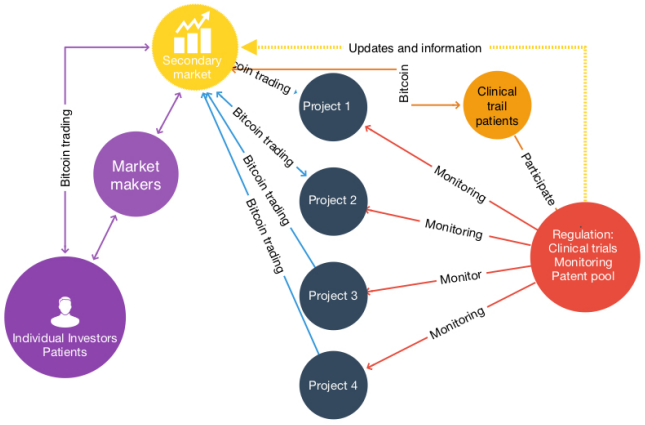


Fig. 3: *Secondary market of crypto-currency.* It brings liquidity by inviting additional market participants, who may have access to better informatics (e.g., systems and synthetic biology) for monitoring, pricing and arbitraging.

been reached. Here we just want to borrow some recent development from the mechanism design literature and present a new funding scheme that can generate better returns to the researchers and investors than the current system.

Smart contracts are important in the context of this principal-agent problem, in which investors (the principal) delegate the researchers (the agents) to search for a solution of the targeted disease. Researchers have private information about the distribution of potential outcome and their own abilities (adverse selection), as well as the efforts they put into researching (moral hazard). They can choose between two different approaches to finish the job: one is a routine approach and the other is an experimental approach. We then combined the findings from the dynamic mechanism design literature (see bibliography for some examples) and propose a new mechanism as follows:

1. At the creation of each SPV (special purpose vehicle, a legal entity with specific responsibilities), researchers submit their research proposals to the mega-fund. This research proposal will include estimates for deadlines, measurable milestones and budget needed for each milestone.
2. All the research proposals are analyzed by the mega-fund and ranked based on the past performance of the researcher, feasibility of the research approach, correlation of this approach to other approaches and its budget. More importantly, the mega-fund may also need to understand the interdependence among the proposals and the nature of coordination, cooperation and competition that it entails.
3. Approved research project is funded under a vesting schedule associated with a cryptocurrency account. A smart contract, imposed on the account, pays the researchers as determined by the research proposal. That is, cryptocurrency for the next stage research is paid to the researcher if and only if the targeted milestone at current stage is met within the predetermined time frame.
4. The budget has an option-like feature: the amount of cryptocurrency paid at each stage is determined by the initial price of the cryptocurrency, or the current price of cryptocurrency if it is lower than the initial price, to meet the proposed budget. In other words, if the price of the cryptocurrency appreciates, then the researchers will enjoy a higher value than his proposed budget; if the price of the cryptocurrency depreciates, then the researcher still gets his proposed budget.
5. For projects that is terminated early because of failing experiments or missing a milestone, the remaining unspent crypto-currencies are redistributed among the other ongoing projects. In this way, the successful projects are rewarded not only by the appreciation of the cryptocurrency, but also by the increasing amount of their cryptocurrency budget. This structure is reminiscent of DARPAs program continuation scheme with hurdles.
6. The results of the research (e.g., drugs) are made available for purchase using crypto currencies to an investor (e.g., a potential patient) upon disease onset. At any time, the patient may also relinquish his rights by selling the crypto currencies in a secondary market.

4 Conclusion

Ina forthcoming paper, Mishra and Qi have simulated the system with realistic parameters and obtained promising results (to be reported in details in the full paper).

At an abstract level, they have demonstrated how to structure the smart contracts in order to simultaneously improve the reputation of and rewards to each researcher, the efficient pricing of drugs via the cryptocurrency, and the liquidity in the resulting market – all made possible by this type of smart contract's ability to address the following information asymmetry problems.

Adverse selection: Since researchers are paid by their long-run performance, their motivation of producing "Lemon projects" are minimized. As in the game-theoretic literature, a "Lemon project" refers to the situation where a researcher has deceptively concealed his lack of skill or the infeasibility of the proposed project by overstating his qualifications or by justifying the project with fraudulent non-reproducible results, respectively. These projects can only meet a few initial milestones and their fund will be shut down as soon as the flaws are detected. Such an outcome will hurt the researchers reputation and significantly reduce their chances of acquiring future research funds from the mega-fund.

Risk Taking: The option setting of the budget ensures that the researchers are protected from the downside risk in research and will be willing to explore the risky non-incremental approaches. First, if any research succeeds in the pool and the price of cryptocurrency appreciates, then the fund for all the other researchers in the pool will also be increased. Therefore the compensation to researchers not only depends on the outcome of his own experiment, but also on the SVPs pool of other experiments. Second, if many research projects in the pool fail, the secondary market may depreciate the cryptocurrency in an irrational way. The mega-fund will guarantee that other researchers project are still properly funded. In this way, the mega-fund ensures that all researchers put proper efforts into their respective projects to avoid a contagious default of the SPV.

Moral Hazard: The design of the fund also ensures that researchers who make a breakthrough in their research will be rewarded proportionately. First, if any compound in the system goes to next phase, the price of the cryptocurrency will jump significantly; second, funds from failed projects will be redistributed among the surviving projects in their funds. Suppose just one single drug gets FDA approval in an SPV, then the team which discovered this drug will get the highest amount of cryptocurrency. It is the same amount of cryptocurrency had the team conducted all the experiments in the SPV on their own. Free Riding: Note that the smart contract has two opposite effects on the researchers motivation. On one hand, the long-run income induced by their reputation motivates them to put as much efforts as they can (career concern); on the other hand, the fund they get is a function of the price of the cryptocurrency, which depends on their performance. However, because the outcome of the mega-fund (or the price of cryptocurrency) is a joint effort of all the researchers in the portfolio, researchers may want to free ride and put less than assumed efforts in research (free ride). Terms and structures of the smart contracts need to be carefully designed so that the career concern motivations dominate the free-ride incentives.

References

1. Qianru Qi and Bud Mishra: Cryptomarket MicroStructure for a Biomedical MegaFund, Under Review, 2017.
2. Esther Kim and Andrew Lo: Business Models to Cure Rare Disease: a Case Study of Solid Biosciences. Journal of Investment Management, 14(4):87101, 2016.
3. David Fagnan, Jose Fernandez, Andrew Lo, and Roger Stein: Can financial engineering cure cancer? In American Economic Review, 103:406411, 2013.
4. X. Yang, Edouardo Debonneuil, Alex Zhavoronkov and Bud Mishra: Cancer megafunds with in silico and in vitro validation: Accelerating cancer drug discovery via financial engineering without financial crisis. Oncotarget, 7(36):5767157678, 2016.

Ghazal: toward truly authoritative web certificates using Ethereum

Seyedehmahsa Moosavi[1] and Jeremy Clark[1]

Concordia University

Abstract. Recently, a number of projects (both from academia and industry) have examined decentralized public key infrastructures (PKI) based on blockchain technology. These projects vary in scope from full-fledged domain name systems accompanied by a PKI to simpler transparency systems that augment the current HTTPS PKI. In this paper, we start by articulating, in a way we have not seen before, why this approach is more than a complementary composition of technologies, but actually a new and useful paradigm for thinking about who is actually authoritative over PKI information in the web certificate model. We then consider what smart contracts could add to the web certificate model, if we move beyond using a blockchain as passive, immutable (subject to consensus) store of data — as is the approach taken by projects like Blockstack. To illustrate the potential, we develop and experiment with an Ethereum-based web certificate model we call Ghazal, discuss different design decisions, and analyze deployment costs.

1 Introductory Remarks

The blockchain data structure and consensus mechanism has received significant interest since being introduced as the underlying technology of the cryptocurrency Bitcoin in Satoshi Nakamoto's (pseudonymous) 2008 whitepaper [25]. In 2014, Buterin presented a new blockchain based application known as Ethereum [10]. As a blockchain-based distributed public network, Ethereum implements a decentralized virtual machine, known as the Ethereum Virtual Machine (EVM), which allows network nodes to execute deployed programmable smart contracts on the Ethereum blockchain [31].This platform enables developers to create and execute blockchain applications called *decentralized applications (dapps)* that are executed correctly according to the consensus of the network. A Dapp's code and data is stored in a decentralized manner on the blockchain. Dapps or smart contracts are now often written in a high level programming language such as *Solidity* which is syntactically similar to Java [1]. Digital smart contracts were first described Nick Szabo in 1993 [28], however they reached a high level of adoption through blockchain technology.

One application of blockchain technology that has received some research and commercial interest is the idea of replacing (or augmenting) the web certificate model used by clients (OS and browsers) to form secure communication channels with web-servers (described in more detail below). This model has been plagued

with issues from fraudulent certificates used to impersonate servers to ineffective revocation mechanisms; see Clark and van Oorschot for a survey [12]. We argue that the application of blockchains to this model is more than an interesting experiment; it is actually a new uni-authoritative paradigm that resolves some of the fundamental issues with the current model — authority and indirection. We also argue that adding programmability to a dapp-based PKI provides benefits beyond using the blockchain as an append-only broadcast channel. Finally, we instantiate our ideas in a novel system called Ghazal implemented in Solidity and deployed on Ethereum. At the time of writing, the overall system costs under $100 to deploy. Basic actions like domain registration costs under $5.

2 Related Work

The HTTPS (HTTP over SSL/TLS) protocol enables secure connections to websites with confidentiality, message integrity, and server authentication. Server authentication relies on a client being able to determine the correct public key for a server. The current web certificate model uses a system of certificate authorities (CAs); businesses that provide this binding in the form of a certificate. Client devices, through the browser and/or the operating system, are pre-installed with a set of known CAs who can delegate authority to intermediary CAs through a protocol involving certificates. When a CA issues a certificate to a web-server, there are generally three types: domain validated (DV) certificates bind a public key only to a domain (*e.g.,* `example.com`), while organization validated (OV) and extended validated (EV) certificates validate additional information about the organization that operates the server (Example, Inc.).

Namecoin is an altcoin (software based on Bitcoin with a distinct blockchain) that implements a decentralized namespace for domain names [17]. The main feature of Namecoin is that for a fee, users can register a `.bit` address and map it to an IP address of their choice. CertCoin [14], and PB-PKI [7] are extensions to Namecoin that add the ability to specify an HTTPS public key certificate for the domain (as well as other PKI operations like expiration and revocation, which we discuss in Section 4.1). Blockstack [6] achieves the same goal by embedding data into a root blockchain, a process called virtualchains that could be instantiated with `OP_RETURN` on Bitcoin's blockchain. These approaches are closest to our own system Ghazal. These systems disintermediate CAs from the web certificate model. The main difference is that we use Ethereum to provide full programmability (motivated below in Section 3.2). In addition, we provide some minor improvements such as allowing multiple keys to be bound to the same domain, as is common for load balancing and CDNs.

Some research has looked at adding transparency, effectively through an efficient log of CA-issued certificates, to augment the current web certificate model. This is a very active area of research that includes certificate transparency (CT) [18], sovereign keys (SKs) [2], and ARPKI [8]. IKP [22] provides an Ethereum-based system for servers to advertise policies about their certificates (akin to a more verbose CAA on a blockchain instead via DNS). Research a bit

further removed from web certificates concerns decentralized PKIs and broader identities. While not decentralized, CONIKS provides a distributed transparency log similar to CT but for public keys (while they could be for anything, email and IM are the primary motivations) [23]. Bonneau provides an Ethereum smart contract for monitoring CONIKS [9]. ClaimChain is similar to CONIKS but finds a middle-ground between a small set of distributed servers (CONIKS) and a fully decentralized but global state (blockchain) by having fully decentralized, local states that can be cross-validated. CONIKS and ClaimChain do not use CAs but rather rely on users validating the logs, which are carefully designed to be non-equivocating. ChainAnchor provides identity and access management for private blockchains [15], while CoSi is a distributed signing authority generic logging [27]. Each of these systems is concerned with logging data (a generic umbrella that encapsulates many of these is Transparency Overlays [11]). As logging systems, they do not provide programmability which is the primary motivation for our system.

Finally, some research has explored having public validated by external parties but replacing the role of CAs with a PGP-style web of trust. SCPKI is an implementation of this idea on Ethereum [5]. Our observation is that for domain validation, a blockchain with a built-in naming system is already authoritative over the namespace and does not require additional validation.

3 Motivation

3.1 Are Blockchains a new paradigm for PKI?

In the related work, most blockchain-approaches to identity (or specifically PKI) motivate their approach with Zooko's triangle; an articulation of three natural properties one might want from an identity system: memorable names, secure (as in hard to impersonate), and a distributed authority for issuing names. His assertion is that two of the three properties can be achieved effortlessly but adding the third is difficult or impossible. Blockchains, starting with Namecoin for domain names and extensions to PKI, are often claimed to resolve this trilemma enabling all three properties in one system. A blockchain is distributed, short human-friendly names can be claimed by anyone, and ownership over a name is secured with a strong cryptographic key.

We approach thinking about this issue a little differently. In the current web certificate model, certificate authorities are meant to be *authorities*: that is, they are authoritative over the namespace they bind keys to. The reality is that the web still runs largely on domain validated certificates [13,16] and for domain validation, certificate authorities are not any more or less authoritative over who owns what domain than you or I. Certificate authorities instead rely on indirection. For example, a certificate authority might validate a request by Alice for a certificate for `alice.com` by sending an email to `admin@alice.com` with a secret nonce that Alice must type into a webform. This involves 2 levels of indirection: (1) CAs appeal to DNS to establish the MX record of the domain

(*i.e.,* the subscriber's mail server's IP address); (2) CAs appeal to SMTP to establish a communication channel to the subscriber. For each level of indirection, there are a set of vulnerabilities which might allow a malicious party to break the verification process and obtain a fraudulent certificate for a domain they do not own. For example, consider the attack surface of email-based validation:

1. **Reserved Emails:** A CA specifies a list of email addresses to receive the challenge. The underlying assumption is that only the domain owner controls this address. However the domain owner might not reserve that email address or even be aware that a certain email address is being used by one of the CAs for this purpose. And recall that just a single CA needs to use a single non-standard email address (*e.g.,* a translation of administrator into their local language) to open up this vulnerability. For example, Microsoft's public webmail service `login.live.com` saw an attacker successfully validate his ownership of the domain using an email address `sslcertificates@live.com` which was open to public registration [32].
2. **Whois Emails:** A CA also optionally draws the email address from the Whois record for the domain. A domain's whois record is generally protected by the username/password set by the domain owner with their registrar. Any attack on this password (*e.g.,* guessing or resetting) or directly on the account (*e.g.,* social engineering [3]) would allow the adversary to specify an email address that they control.
3. **MX Record:** A CA establishes the IP address of the mailserver from the MX record for the domain. As above, all domain records including the MX record is managed through the owner's account with her domain registrar. Any method for obtaining unauthorized access to this account would enable an adversary to list their own server in the MX record and receive the email from the CA.
4. **DNS Records:** If an adversary cannot directly change a DNS record, they might conduct other attacks on the CA's view of DNS. For example, they might employ DNS cache poisoning which can result in invalid DNS resolution [26]. They might also exploit an available dangling DNS record (Dare) [19]. Dares occur when data in a DNS record (such as CNAME, A, or MX) becomes invalid but is not removed by the domain owner. For example, if the domain owner forgets to remove the MX record (the IP address of the server) from DNS, the associated DNS MX record is said to be dangling. If an adversary can acquire this IP address at some future point, he is able to redirect all traffic intended for the original domain to his server, including information sufficient for a CA's domain validation process. Thus a malicious party can use a Dare to obtain a fraudulent certificate. In a uni-authoritative system, Dares are still possible (old data that hasn not been purged from the system) but the public keys dangle with the IP address, which resolves the security issue for mis-issued certificates
5. **SMTP:** Once the CA establishes the mailserver's record, it will send the email to the mailserver with SMTP (the standard protocol for transfer of email). Since the email contains a secret nonce, confidentiality of this email

is crucial. SMTP uses opportunistic encryption that is not secure against an active adversary. Thus a man-in-the-middle between the CA's mailserver and the ultimate destination (including an forwarding mailservers) could request a fraudulent certificate, intercept the ensuing email, reply with the correct nonce, and be issued the fraudulent certificate.

6. **Email Accounts:** Email accounts are generally protected with a username and password (over IMAP or POP3) to prevent unauthorized access. In some cases, they might be protected with a client certificate. An adversary who can gain access to any one of the accounts that should be reserved by the domain owner (*e.g.,* textttadmin, `hostmaster`, `webmaster`, *etc.*) could obtain a fraudulent certificate for that site. This could include guessing or resetting the password, using social engineering, or obtaining access to the server hosting the email for the account.

Blockchains are actually a new paradigm; they collapse the indirection for domain validation. If a PKI were added to a blockchain, who would be authoritative over the namespace of domain names? When domain names themselves are issued through the blockchain (*e.g.,* Namecoin), then the blockchain is actually the authoritative entity. Arguably, this indirection can be collapsed in the traditional web certificate model as well. There DNS (in conjunction with ICANN) is authoritative over the namespace and if ICANN/DNS held key bindings, there would be no indirection or CAs needed — indeed, this is exactly the proposal of DANE. Thus blockchains and DANE are both examples of what we might call a uni-authoritative paradigm. A deployment issue with DANE is that DNS records do not generally have message integrity (except via the under-deployed DNSSEC) whereas blockchain transactions do.

3.2 Does programmability add anything?

In the related work, some systems take a uni-authoritative approach while others rely on third party authorities (generally, CAs or web of trust). Most systems that use a blockchain (or similar trasnparency log) do it in a passive way—as an append-only broadcast channel; a few systems actually use smart contracts or the programmability that a blockchain provides. Of all these systems, to the best of our knowledge, none are both uni-authoritative and use programmability. We have argued the merits of uni-authoriatiative above, what about programmability? What does it provide?

Programmability, or PKI bindings within a smart contract, can enable features that seem desirable. A few examples include: external contracts that can easily obtain information about a domain in making decisions; atomonicity within domain name transfers where payments and transfers are inputs to the same transaction (*e.g.,* even Namecoin relies on a third party tool called ANTPY to perform atomic name ownership transfer transactions); and fancier options for transfering domain names: we implement an auction where any domain owner can auction off their domain within the smart contract itself. The reader might think of other features that programmability could add.

In defence of non-programmable blockchain-based PKIs, such as Blockstack, it is not clear how well a system like ours scales and what demands it puts on user clients to quickly fetch information on domains. We return to this in the next section, however we note here that we are not claiming programmability necessarily wins out in the end, only that it is worth exploring from a research perspective to better understand the trade-offs.

4 The Ghazal System

Our proposed scheme is entitled Ghazal, a smart contract-based naming and PKI uni-authoritative system. [1] It enables entities, whether they are people or organizations, to fully manage and maintain control of their domain name without relying on trusted third parties. In Ghazal, a user can register an unclaimed domain name as a globally readable identifier on the Ethereum blockchain. Subsequently, she is able to assign arbitrary data, such as TLS certificates to her domain. These values are globally readable, non-equivocating, and not vulnerable to the indirection attacks outlined above. The penalty paid for a uni-authoritative approach is that Ghazal has to carve out its own namespace that is not already in use (*e.g.,* names ending in `.ghazal` like Namecoin's `.bit` or Blockstack's `.id`). OS and browsers would have to be modified before any system like this can be used. Anyone can claim a domain on a first-come, first-serve basis. Because it is decentralized, names cannot be re-assigned without the cooperation of the owner (whereas an ICANN address like `davidduchovny.com` can be re-assigned through adminstrative mediation).

The design of Ghazal consists of two essential elements. First, the smart contract that resides on the Ethereum blockchain and serves as an interface between entities and the underlying blockchain. The second primary component of the system are the clients, including people or organizations that interact with Ghazal smart contract in order to manage their domain names. Figure 1 represents the primary states a domain name can be in and how state transitions work. These states are enforced within the code itself to help mitigate software security issues related to unintended execution paths.

4.1 Exploring Ghazal design choices

Beyond simply presenting our design, we think it is useful to explore the landscape of possible designs. To this end, we discuss some deployment issues that we faced where there was no obvious "one right answer." These are likely to be faced by others working in this space (whether working narrowly on PKI or broad identity on blockchain solutions).

Design Decision #1: Domain Name Expiration
Typically domain name ownership eventually *expires*. Once a domain expires,

[1] `https://github.com/mahsamoosavi/Ghazal`

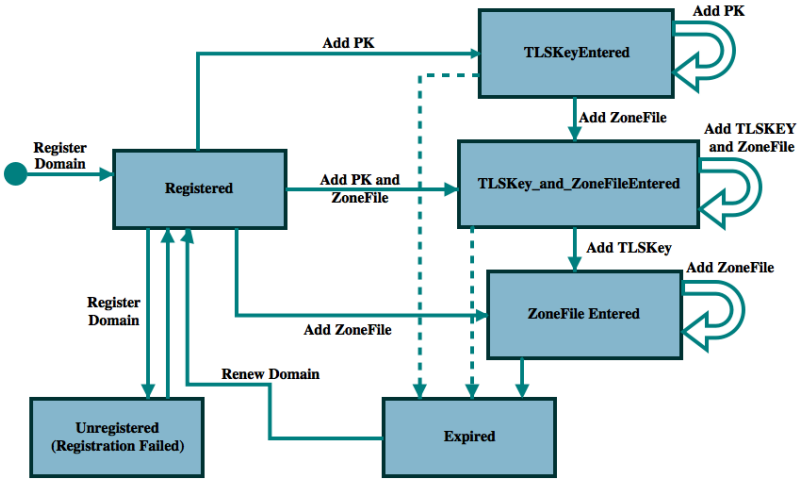


Fig. 1: Primary states and transitions for a domain name in Ghazal.

it is returned to the primary market, except if the users renews it. However, expiration does not necessarily have to mean a disclaimer of ownership; there are other options.

1. **Domain names never expire and last forever.** Designing a system with no domain name expiration would be highly vulnerable to domain squatting. Domain squatting is registering domain names in speculation that the will increase in value. These domain names generally do not point to any relevant IP address (except to earn revenue on accidental visits). If domain names never expire, squatting may be significantly problematic as squatted names would be locked forever while legitimate users will end up choosing unusual names from the remaining namespace. To be clear, even without expiration, if domains are cheap, squatting is problematic (*e.g.,* Namecoin [17]).
2. **Domain names get deleted once they expire, except being renewed by the user.** The most restrictive system design is where a domain name effectively gets deleted and is returned to the registry of unclaimed names once it expires, unless the user renews it. This model has the following two issues. First, if a browser tries to resolve an expired domain, because the blockchain has a complete, immutable history of that domain, we would expect users to want it resolved according to the previous owner. Rolling back expirations is possible in a way not supported by DNS and it resolves simple human errors of forgetting to renew domains, so we do not expect browsers to necessarily fail when it could make a sensible guess as to which server their users are looking for. The second reason to drop the deletion model of expiration is that Ethereum contracts can only run when a function is called. If no one calls a function at expiration time, the contract cannot self-execute to modify itself. The fact that it is expired can be inferred from contract if it

includes a time but the contract itself will not transition states until someone calls a function that touches that particular contract. An alternative is to rely on a third party like Ethereum Alarm Clock [4] for scheduling future function calls. This is suitable only if the threat model permits relying on a trusted third party and a single point of failure (for this one feature).

3. **Control over domain names is lost once they expire, except being renewed by user.** In Ghazal, expired domains continue to function although the owner (i) looses the sole claim to that domain and cannot preserve it if someone else purchases it, and (ii) she cannot modify the domain in anyway (*e.g.,* add certificates or change zone information) unless if she first renews it. Essentially, purchasing a domain name does not entitle an entity to own it forever; expired domain names are returned back to the primary market and are available for all the users within the system. However since a full history of a domain is present, the system's best effort at resolving the domain will be to preserve the last known state. Expiration in conjunction to the amount of the fee will influence the degree of domain squatting, and having expiration at all will allow abandoned domains to churn if they are under demand.

Design Decision #2: Registration Fees

In Ghazal, new registrations and renewals require a fee. This fee is a deterrent against domain squatting. The fee amount is difficult to set and no fee will be perfectly priced to be exactly too high for squatters but low enough for all 'legitimate' users. Rather it will trade-off the number of squatters with the number of would-be legitimate users who cannot pay the fee. Namecoin is evidently too cheap and ICAAN rates seem reasonable. We leave this as a free parameter of the system. The important decisions are: (1) in what currency are they paid and (2) to whom. Every Ethereum-based system, even without a fee, will at least require gas costs. Additional fees could be paid in Ether or in some system-specific token. Since it is a decentralized system and the fee is not subsidizing the efforts of any entity involved, there is no one in particular to pay. The fee could be paid to an arbitrary entity (the system designer or a charity), burned (made unrecoverable), or to the miners. In Ghazal, fees are paid in Ether and are released to the miner that includes the transaction in the blockchain.

Design Decision #3: Domain Name Renewal

We design Ghazal in such a way that the domain owners can renew their domains before their validity period comes to an end, however they cannot renew an arbitrary number of times. Specifically, a renewal period becomes active after the domain is past 3/4 of its validity period. Renewal pushes the expiration time forward by one addition of the validity period (thus renewing at the start or end of the renewal period is inconsequential and results in the domain having the same expiration time). Requiring renewal keeps users returning regularly to maintain domains, and unused domains naturally churn within the system. Domain name redemption period can take different values. We experiment with

```
1  //Possible states of every auction.
2  enum Stages {Opened, Locked, Ended}
3
4      struct AuctionStruct
5      {  uint CreationTime;
6          address Owner;
7          uint highestBid;
8          address highestBidder;
9          address Winner;
10         Stages stage;
11         //To return the bids that were overbid.
12         mapping(address => uint) pendingReturns;
13         //To return the deposits the bidders made.
14         mapping(address => uint) deposits;
15         //Once an address bids in the auction, its associated boolean value
               will be set to true within the "already_bid" mapping.
16         mapping(address => bool) already_bid;
17         bool AuctionisValue;
18       }
19 //AuctionLists mappings store AuctionStructs.
20 mapping (bytes32 => AuctionStruct) internal AuctionLists;
```

Code 1.1: Implementation of AuctionStruct and AuctionLists mapping in Ghazal* smart contract.

a validity period of 1 year; thus, the renewal period would start after 9 months and last 3 months.

Design Decision #4: Domain Name Ownership Transfer

In Ghazal, domain owners can transfer the ownership of their unexpired domains to new entities within the system. Basically, transferring a domain name at the Ethereum level means changing the address of the Ethereum account that controls the domain. Our system offers two ways of transferring the ownership of a domain:

1. **Auctioning off the domain name.** A domain owner can voluntarily auction off an unexpired domain. Once an auction is over, the domain is transferred to the highest bidder, the payment goes to the previous owner of the domain, and the validity period is unaffected by the transfer (to prevent people from shortcutting renewal fees by selling to themselves for less than the fee). If there are no bidders or if the bids do not reach a reserve value, the domain is returned to the original owner. While under auction, a domain can be modified as normal but transfers and auctions are not permitted. To implement the auction feature, we use the fact that Solidity is object-oriented. We first deploy a basic Ghazal function without advanced features like auctions, and then use *inheritance* to create a child contract Ghazal* that adds the auction process. Using Ghazal*, a user can run any number of auctions on any number of domains he owns. This is implemented through a mapping data structure called *AuctionLists* to store every auctions along with

its attributes. *AuctionLists* accepts *Domain names* as its keys, and the *AuctionStructs* as the values (see Code 1.1). Using the mapping and Ethereum state machine, we enforce rules to prevent malicious behaviours *e.g.,* domain owners can auction off a domain only if there is no other auction running on the same domain. To encourage winners to pay, all bidders must deposit a bounty in Ether the first time they bid in an auction (amount set by the seller). This is refunded to the losers after bidding closes, and to the winner after paying for the domain. Without this, users might disrupt an auction by submitting high bids with no intention of paying.

```
modifier CheckDomainExpiry(bytes32 _DomainName) {
       if (Domains[_DomainName].isValue == false)
          {Domains[_DomainName].state=States.Unregistered;}
       if (now>=Domains[_DomainName].RegistrationTime+10 minutes)
          {Domains[_DomainName].state = States.Expired;}
        _;
    }
modifier Not_AtStage(bytes32 _DomainName, States stage_1, States stage_2)
     {
        require (Domains[_DomainName].state != stage_1 && Domains[
            _DomainName].state != stage_2);
        _;
    }
modifier OnlyOwner(bytes32 _DomainName) {
        require(Domains[_DomainName].DomainOwner == msg.sender);
        _;
    }
function Transfer_Domain(string _DomainName,address _Reciever,bytes32
    _TLSKey,bytes32 _Zone) public
CheckDomainExpiry(stringToBytes32(_DomainName))
Not_AtStage(stringToBytes32(_DomainName),States.Unregistered,States.
    Expired)
OnlyOwner(stringToBytes32(_DomainName))
    {
        DomainName = stringToBytes32(_DomainName);
        Domains[DomainName].DomainOwner = _Reciever;
        if (_TLSKey == 0 && _Zone != 0) { Wipe_TLSKeys(DomainName); }
        if (_Zone == 0 && _TLSKey != 0 ) { Wipe_Zone(DomainName); }
        if (_Zone == 0 && _TLSKey == 0 ) { Wipe_TLSKeys_and_Zone(
            DomainName); }
    }
```

Code 1.2: `Transfer_Domain` function of Ghazal smart contract.

2. **Transfer the ownership of a domain name.** A domain owner can also transfer an unexpired domain to the new Ethereum account by calling the *Transfer_Domain* function which simply changes the Ethereum address that controls the domain name. The owners can also decide to either transfer domain's associated attributes (*e.g.,* TLS certificates) or not, when they transfer the domain. This is possibe with either supplying these attributes with zero or other desired values when calling the `Transfer_Domain` function (see Code 1.2).

To prevent from MITM attacks, TLS certificates should be revoked once a domain name is transferred. However, security incidents reveal that this is not

commonly enforced in the current PKI. For instance, Facebook acquired the domain `fb.com` for $8.5M in 2010, yet no one can be assured if that the previous owner does not have a valid unexpired certificate bound to this domain [12]. This has been successfully enforced in our system as the new owner of the domain is capable of modifying the domain's associated TLS keys, which results in protecting communications between the clients and his server from eavesdropping.

Design Decision #5: Toward Lightweight Certificate Revocation

In the broader PKI literature, there are four traditional approaches to revocation [24]: certificate revocation lists, online certificate status checking, trusted directories, and short-lived certificates. Revocation in the web certificate model is not effective. It was built initially with revocation lists and status checking, but the difficulty of routinely obtaining lists and the frequent unavailability of responders led to browsers failing open when revocation could not be checked. Some browsers build in revocation lists, but are limited in scope; EV certificates have stricter requirements; and some research has suggested deploying short-lived certificates (*e.g.,* four days) that requires the certificate holders to frequently renew them [29] (in this case, certificates are not explicitly revoked, they are just not renewed). Which model does a blockchain implement? At first glance, most blockchain implementations would implement a trusted directory: that is, a public key binding is valid as long as it is present and revocation simply removes it. The issue with this approach on a blockchain is how users establish they have the most recent state. With the most recent state in hand, revocation status can be checked. This check is potentially more efficient than downloading the entire blockchain (this functionality exists for Bitcoin where it is called SPV and is a work in progress for Ethereum where it is called LES). However a malicious LES server can always forward the state immediately preceding a revocation action and the client cannot easily validate it is being deceived.

At a foundational level, most revocation uses a `permit-override` approach where the default state is permissive and an explicit action (revocation) is required. Short-lived certificates (and a closely related approach of stapling a CA-signed certificate status to a certificate) are `deny-override` meaning the default position is to assume a certificate is revoked unless if there is positive proof it is not. This latter approach is better for lightweight blockchain clients as LES servers can always lie through omitting data, but cannot lie by including fraudulent data (without expending considerable computational work). As an alternative or compliment, clients could also take the consensus of several LES servers, although this 'multi-path probing' approach has some performance penalties (it has been suggested within the web certificate model as Perspectives [30] and Convergence [21]).

In Ghazal, public keys that are added to a domain name expire after a maximum lifetime, *e.g.,* four days. Expiration is not an explicit change of state but is inferred from the most recent renewal time. Owners need to rerun the key binding function every several days to renew this. If an owner wants to revoke a key, she simply fails to renew. To verify the validity of a certificate, one is now

Operation	Gas	Gas Cost in Ether	Gas Cost in USD
Register	169 990	3.56×10^{-3}	$3.15
Renew	54 545	1.14×10^{-3}	$1.01
Transfer_Domain	53 160	1.11×10^{-3}	$0.98
Add_TLSKey	77 625	1.63×10^{-3}	$1.43
Add_ZoneFile	57 141	1.19×10^{-3}	$1.05
Add_TLSKey_AND_ZoneFile	68 196	1.43×10^{-3}	$1.26
Revoke_TLSkey	37 672	7.91×10^{-4}	$0.69
StartAuction	119 310	2.50×10^{-3}	$2.21
Bid	112 491	2.36×10^{-3}	$2.08
Withdraw_bids	46 307	9.72×10^{-4}	$0.85
Withdraw_deposits	47 037	9.87×10^{-4}	$0.87
Settle	77 709	1.63×10^{-3}	$1.44
Ghazal* Contract Creation	2 402 563	0.05	$44.54

Table 1: Gas used for operations in the Ghazal* smart contract.

able to use a LES-esque protocol. Once a user queries a semi-trusted LES node for a corresponding record of a domain, the node can either return a public key that is four days old, which user will assume is revoked, or a record that newer that the user will assume is not revoked. Although this approach requires the frequent renewal of public keys, it is a cost that scales in the number of domains as opposed to revocation checks which scale in the number of users accesses a domain.

5 Evaluation

The aim of this section is to provide the technical implementation details of our system on the Ethereum blockchain. We specifically discuss the costs related to the deployment of Ghazal* smart contract on the Ethereum blockchain in addition to executing its functions on the Ethereum virtual machine. Moreover, a smart contract analysis tool is used to analyze the security of our system against a several number of security threats to which smart contracts are often vulnerable.

5.1 Costs

Ghazal smart contract is implemented in 370 lines of Solidity language, a high level programming language resembles to JavaScript, and tested on the Ethereum test network. We use the Solidity compiler to evaluate the rough cost for publishing the Ghazal* smart contract on the Ethereum blockchain as well as the cost for the various operations to be executed on the Ethereum virtual machine. As of January 2018, 1 gas = 21×10^{-9} ether[2], and 1 ether = $882.92[3].

[2] https://ethstats.net/
[3] https://coinmarketcap.com/

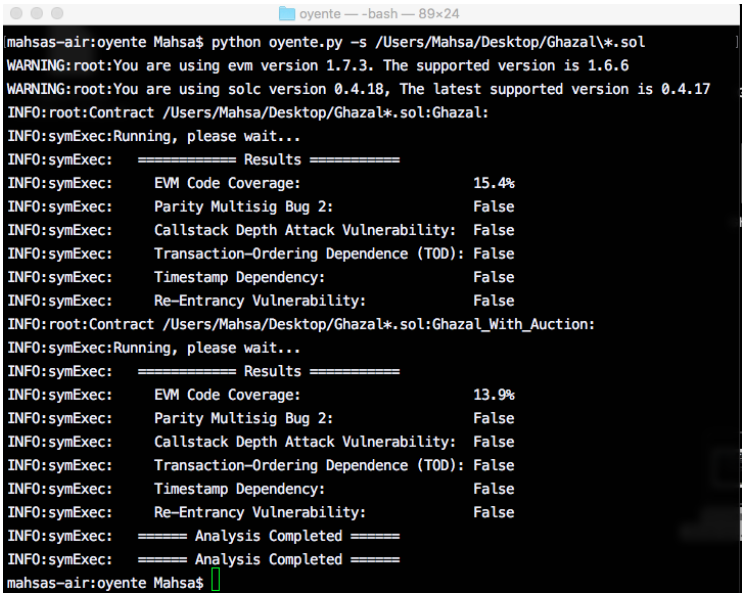


Fig. 2: Results of Ghazal* security analysis using Oyente [20].

Table 1 represents the estimated costs for Ghazal* (and its inherited Ghazal functionality) smart contract deployment and function invocation in both gas and USD. As it can be seen from both Table 1, the most considerable cost is the one-time cost paid to deploy the system on Ethereum. There are then relatively small costs associated with executing the functions, *i.e.,* users could easily register a domain by paying $3.15 or they could bind a key to the domain they own for a cost of $1.43, which is relatively cheap when compared with the real world costs associated with these operations.

5.2 Security Analysis

Ethereum smart contracts, in particular the ones implemented in Solidity, are notorious for programming pitfalls. As they generally transfer and handle assets of considerable value, bugs in Solidity code could result in serious vulnerabilities which can be exploited by adversaries. We use standard defensive programming approaches, in particular around functions that transfer money (such as the auction function that refunds the security deposits), by using explicitly coded state machines and locks, and by not making state-changes after transfers. We also analyze Ghazal and Ghazal* against Oyente, a symbolic execution tool proposed by Luu *et al.* [20] which looks for potential security bugs like the re-entry attack (infamously). The results of the security analysis represent that both of the smart contracts are not vulnerable to any known critical security issue (see Figure 2).

6 Concluding Remarks

We hope that uni-authoritative systems with programmability continue to be explored in the literature. There are many open problems to work on. First and foremost is understanding the scalability issues and how to minimize the amount of data a client browser needs to fetch for each domain lookup. Blockstack has done an excellent job on this issue for non-programmable contracts. Future work could also look at the layer above the smart contract: building web tools with user interfaces to enable interaction with the underlying functions. Finally, while auctions are one illustrative example of why programmability might be added to a PKI, we are sure there are many others. The modular design of Ghazal using object-oriented programming should allow easy additions to our base contract, which we will provide as open source. Indeed, the auction itself in Ghazal* was added via inheritance and one function override (to enforce that ownership transfers, part of the parent class, could not be called during a live auction).

References

1. Ethereum development tutorial ethereum/wiki wiki. https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial. (Accessed on 07/12/2017).
2. git.eff.org git - sovereign-keys.git/blob - sovereign-key-design.txt. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD. (Accessed on 01/10/2018).
3. Godaddy owns up to role in epic twitter account hijacking — pcworld. https://www.pcworld.com/article/2093100/godaddy-owns-up-to-role-in-twitter-account-hijacking-incident.html. (Accessed on 02/13/2018).
4. Home. http://www.ethereum-alarm-clock.com/. (Accessed on 12/29/2017).
5. M. Al-Bassam. Scpki: A smart contract-based pki and identity system. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 35–40. ACM, 2017.
6. M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman. Blockstack: A global naming and storage system secured by blockchains. In *USENIX Annual Technical Conference*, pages 181–194, 2016.
7. L. Axon and M. Goldsmith. Pb-pki: a privacy-aware blockchain-based pki. 2016.
8. D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. Arpki: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393. ACM, 2014.
9. J. Bonneau. Ethiks: Using ethereum to audit a coniks key transparency log. In *International Conference on Financial Cryptography and Data Security*, pages 95–105. Springer, 2016.
10. V. Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.
11. M. Chase and S. Meiklejohn. Transparency overlays and applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 168–179. ACM, 2016.

12. J. Clark and P. v. Oorschot. SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE S&P*, 2013.
13. Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the https certificate ecosystem. In *IMC*, 2013.
14. C. Fromknecht, D. Velicanu, and S. Yakoubov. Certcoin: A namecoin based decentralized authentication system 6.857 class project. 2014.
15. T. Hardjono and A. S. Pentland. Verifiable anonymous identities and access control in permissioned blockchains.
16. R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: A thorough analysis of the X.509 PKI using active and passive measurements. In *IMC*, 2011.
17. H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*, 2015.
18. B. Laurie. Certificate transparency. *Queue*, 12(8):10, 2014.
19. D. Liu, S. Hao, and H. Wang. All your dns records point to us: Understanding the security threats of dangling dns records. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1414–1425. ACM, 2016.
20. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.
21. M. Marlinspike. SSL and the future of authenticity. In *Black Hat USA*, 2011.
22. S. Matsumoto and R. M. Reischuk. Ikp: Turning a pki around with blockchains. *IACR Cryptology ePrint Archive*, 2016:1018, 2016.
23. M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman. Coniks: Bringing key transparency to end users. In *USENIX Security Symposium*, pages 383–398, 2015.
24. M. Myers. Revocatoin: Options and challenges. In *Financial Cryptography*, pages 165–171. Springer, 1998.
25. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
26. S. Son and V. Shmatikov. The hitchhikers guide to dns cache poisoning. *Security and Privacy in Communication Networks*, pages 466–483, 2010.
27. E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 526–545. Ieee, 2016.
28. N. Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.
29. E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh. Towards short-lived certificates. *Web 2.0 Security and Privacy*, 2012.
30. D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Annual Tech*, 2008.
31. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.
32. M. Zusman. Criminal charges are not pursued: Hacking pki. *DEFCON 17*, 2009.

The Game among Bribers in a Smart Contract System

Abstract. Blockchain has been used to build various applications, and the introduction of smart contracts further extends its impacts. Most of existing works consider the positive usage of smart contracts but ignore the other side of it: smart contracts can be used in a destructive way, particularly, they can be utilized to carry out bribery. The hardness of tracing a briber in a blockchain system may even motivate bribers. Furthermore, an adversary can utilize bribery smart contracts to influence the execution results of other smart contracts in the same system. To better understand this threat, we propose a formal framework to analyze bribery in the smart contract system using game theory. We give a full characterization on how the bribery budget of a briber may influence the execution of a smart contract if the briber tries to manipulate its execution result by bribing users in the system.

1 Introduction

Various applications are developed on top of blockchain technology [32, 34, 33]. However, most of these works assume that the blockchain is a perfect system, e.g., all records stored in the system are correct, and ignore the complexity of the way that the decentralized system achieves consensus. For purely cryptocurrency systems, both static model [24] and game theory model [20, 28] have been used to analyze their security features. The introduction of the smart contract makes the situation trickier while extending the applicability of blockchain technology. A smart contract can involve multiple users/participants and have a high value stake. Thus, it has the potential to be more critical than mining in pure cryptocurrency systems (e.g., Bitcoin), in which only a fixed reward is paid to successful miners. The amount of cryptocurrency involved in a contract may be many times and significantly higher than the cost of running the contract itself. Therefore, users involved in a smart contract have the incentive to push through a certain outcome. In particular, they may achieve such a goal through bribery, i.e., offering cryptocurrencies to other users in the system. Interestingly, bribery itself can also be carried out using smart contracts. A recent work discussed this concept and proposed a straightforward framework to implement bribery on blockchain [19] where the briber offers incentive to the bribee through a smart contract.

Bribery is a serious problem as it may help to compromise the fundamental assumption of smart contract execution model based on consensus or majority accepted outcome. Note that a user is honest in mining does not necessarily means that he/she will remain honest when offered with monetary reward in

making decisions. Their honesty is even more questionable when taking into consideration the unlinkability of users' identities to real persons, and the fact that there is no punishment for reporting a wrong execution result in many smart contract systems like Ethereum. Therefore, it is important to investigate the problem whether a briber can succeed in manipulating a smart contract execution result.

It is remarkable that the execution of a smart contract can be cast as an election and we may leverage the research on elections to understand the bribery problem in a smart contract system. Specifically, we can view users in the system as voters, and all the possible outcome of a smart contract as candidates. Each voter (user) will vote for a specific candidate (outcome), and a briber will bribe voters to alter the election result (smart contract execution outcome). We remark that by using an election model we are actually simplifying the consensus protocol implemented in a smart contract system without considering, e.g., the Byzantine behavior of a user who tries to send different messages to different other users. However, note that such kind of behaviors typically influence users who are following the protocol. In this paper, we take a game theoretical point of view by treating all users as rational people who are trying to maximize their own profit, and will therefore stick to the choice which is the best for their own interest regardless of the choices of others. Hence, it is reasonable to adopt an election model.

There exist a series of papers focusing on the bribery problem in an election model, see, .e.g., [14, 2, 36, 8, 26, 35, 17, 4, 11, 3, 23, 18, 9]. Specifically, researchers have studied extensively the computational complexity of the bribery problem and show that in many settings it is NP-hard for a briber to decide which subset of voters should he/she bribe (see, e.g., [16] for a nice survey). Such hardness results can also be viewed as a way to discourage people from carrying out bribery, if computational complexity is of concern.

Classical hardness results for the election model apply readily to the bribery problem in a blockchain system by viewing a smart contract execution as an election. However, we observe that a briber needs to overcome more difficulties if he/she really wants to carry out bribery in a blockchain system. Indeed, a briber not only needs to handle the computational complexity in determining a suitable subset of voters to be bribed, but he/she may also have to compete with other bribers in the system. Note that in most real-world elections, bribery is carried out in secrecy. A person, once offered a bribe, may either take it and cast his/her vote shortly afterwards, or reject it. The "incorrectness" in the nature of bribery prevents it from becoming a free market where bribers "sell" their bribes to people. However, things change completely in a blockchain system. As we will provide details in the following section, a briber is able to establish a smart contract with a bribee. The smart contract will be executed by users in the system and a transfer of cryptocurrencies will be carried out once the contract is fulfilled, i.e., once the bribee casts his/her vote accordingly. In this case, a bribee may establish smart contracts with multiple bribers and strategically chooses the best. The unlinkability from a user identity in a blockchain system

to a real person behind and the fact that a smart contract may not necessarily be executed immediately allow a user to easily involve in multiple smart contracts. Such a situation poses a severe task to bribers and they end up in competing with each other unavoidably without even knowing their opponents. Under such a competition in a blockchain system, how difficult it is for a specific briber to win? This paper is targeting at such a problem.

Our contributions. There are two major contributions of this paper. First, we study the bribery problem in a blockchain system from a game theoretical point of view and model it as a smart contract bribery game. This is a first step towards a better understanding of the bribery problem in a blockchain system; and may also be of separate interest to the studies of elections. In this model, every briber is a player and has a bribing budget which can be allocated to voters. Every voter has a bribing price p_j. The voter will only take smart contracts that offer a price no less than p_j. Once he/she is offered multiple smart contracts, he/she will fulfill the one with the highest price (ties are broken arbitrarily). The strategy set of a briber is all possible allocations of the budget to voters.

Second, given a smart contract bribery game, we consider its Nash equilibrium. We are particularly interested in the following problem: if a briber is very lucky, can he/she compromise the smart contract execution by getting the majority of votes through a small amount of budget? The answer is no. We show that, a briber cannot win more than 50% of the votes unless he/she controls more than 20% of the total bribing budgets in *any* Nash equilibrium. That is, even if the briber is lucky enough to end up in a Nash equilibrium that is the best for him/her, he/she still needs to have a significantly large bribery budget, more than 20% of the sum of all the budgets, in order to manipulate the execution result arbitrarily.

Organization of the paper. The remainder of the paper is organized as follows: In Section 2 we give a short review of smart contract and describe the problem we address in this paper. In Section 3 we present our main result by studying the Nash equilibria of the smart contract bribery game. In Section 4 we give further discussion on our results. Section 5 discusses related work, and we conclude the paper in Section 6.

2 Preliminaries and Problem Statement

Smart contract We begin by defining smart contracts. The definition provided by Szabo in 1997 is [29]:

Definition 1. *A smart contract is a set of promises, specified in a digital form, including protocols within which the parties perform on these promises.*

A blockchain system, equipped with smart contracts, is a powerful tool that allows users to build various applications on top. In particular, a voting system can be implemented on blockchain. We first briefly describe the election model for a voting system studied in the literature.

Election model In an election, given are a set of n candidates $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ and a set of m voters $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$. Each voter V_j has a preference list of candidates, which is essentially a permutation of candidates, denoted as τ_j. The preference of v_j is denoted by $(C_{\tau_j(1)}, C_{\tau_j(2)}, \ldots, C_{\tau_j(m)})$, meaning that v_j prefers candidate $C_{\tau_j(z)}$ to $C_{\tau_j(z+1)}$, where $z = 1, 2, \ldots, m-1$.

An *election rule* is implemented, which takes as input the set of candidates and voters together with their preference lists, and outputs a set of winner(s). There are various election rules studied in the literature. In this paper, we focus on one of the most fundamental rules called *plurality*. In plurality, every voter votes for exactly one candidate which is on top of his/her preference list. The candidate(s) with the highest number of votes then become the winner(s).

The abstract election model is general enough to incorporate a lot of real-world elections as well as other applications that involve voting in their execution. In particular, it is very much relevant to a blockchain system since almost all decisions made in such a system, e.g., block construction and verification [31], are based on the consensus among users. A consensus protocol can be modeled as an election where every user votes for his/her decisions, and eventually one decision is elected by the system.

Bribery in an election In recent years, the problem of bribery in an election has received much attention in the literature [14, 2, 36, 8, 26, 35, 17, 4, 11, 3, 23, 18, 9]. On a high level, bribery in an election is defined as a way to manipulate the election by giving monetary reward to voters so as to change their preference lists. Researchers have proposed different bribery models. In this paper, we focus on the *constructive bribery model*, that is, the briber tries to make one specific candidate become the winner by bribing a subset of voters. This is particularly the case when bribery happens in a blockchain based system – a briber tries to make the system to reach a specific consensus.

Bribery through smart contract In most real-world elections, briberies are carried out in secrecy. It is, however, interesting that briberies can be carried out "publicly" using smart contracts. Roughly speaking, the briber and the user to be bribed (or bribee) can create a special smart contract that claims a transfer of cryptocurrency upon the condition that the user votes for a specific candidate (decision). Users of the system will execute this smart contract. Once the condition is satisfied, the transfer of the cryptocurrency will be enforced by the system. The anonymous feature of a blockchain system, especially the unlinkability of a user account from the real person behind, allows part of the information of the bribery to be transparent, e.g., the transfer of cryptocurrency from one account to another, while preserves the privacy of the persons involved.

The concept of carrying out bribery through smart contract naturally follows from many real-world contracts that are created to facilitate bribery. However, there is a lack of a systematic study on the creation and execution of such smart contracts for bribery, and its influence on the whole blockchain system. A very recent paper by Kothapalli and Cordi [19] gave the first detailed study on the creation and execution of the smart contracts for bribery and presented pseudo

codes. Briefly speaking, the whole bribery procedure, via smart contracts, is divided into three phases: (i) Propose stage. The briber creates a briber contract indicates the incentive that the bribee will receive upon fulfilling the bribe and/or the punishment if the bribee fails to fulfill that. The contract is submitted to the blockchain. (ii) Commit stage. A bribee who decides to participate creates a claim on the blockchain. (iii) Verify stage. After a time period, if the bribe condition is reached, the bribee can get the incentive. Otherwise, the bribee pays the penalty.

Given their research [19], it becomes crucial to understand the impact of such smart contracts for bribery to the whole blockchain system. Although we may leverage the research on bribery in elections, the problem of bribery via smart contracts has its own unique characteristics. Particularly, when there are multiple bribers in the system, the bribee is free to participate in any smart contract for bribery and he/she can thus strategically maximize his/her own profit. In this paper, we try to understand the behavior of bribers and bribees through game theory. Towards this, we first introduce some basic concepts.

Definition 2 ([25]). *A* normal form game Γ consists of:

- A finite set N of players (agents).
- A nonempty set Q_i of strategies available for each player $i \in N$.
- A preference relation $\preceq_i$ on $Q = \times_{j \in N} Q_j$ for each player i.

We restrict our attention to normal form games in this paper. For simplicity, when we say a game, we mean a normal form game. We consider Nash equilibrium in this paper. A Nash equilibrium is a solution concept of a game involving two or more players in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by unilaterally changing his/her own strategy [25].

Taking a game theoretical point of view, we are able to model the bribery problem in a blockchain system with multiple bribers as follows.

Smart contract bribery game We first describe the basic setting for the smart contract bribery game. Given are a set of n candidates $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, a set of m voters $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ and a set of k bribers $\mathcal{B} = \{B_1, B_2, \cdots, B_k\}$. Each briber B_h has a budget b_h for bribing and prefers one specific candidate. Each voter v_j has a preference list τ_j and a bribing price p_j. Each briber can sign a smart contract with a voter, which offers a certain amount of reward in cryptocurrency if the voter changes his/her preference list and votes for the candidate preferred by the briber. A voter v_j can sign a smart contract with every briber and then do the following:

- he/she will discard all smart contracts that offer a price lower than p_j;
- if there are multiple smart contracts offering a price larger than p_j, he/she will pick the one with the highest price and vote for the candidate preferred by this briber;

– ties are broken arbitrarily, i.e., the voter will randomly choose one smart contract if there are several smart contracts offering the same highest price (larger than or equal to p_j).

Note that if all the smart contracts are offering a price lower than p_j, the voter will vote honestly.

Bribers and the candidates need not be the same, however, as each briber prefers a distinct candidate, we assume for simplicity that the briber is *the same as* the candidate he/she prefers, i.e., $\mathcal{B}$ is a subset of the candidates. By re-indexing the candidates, we may assume without loss of generality that $B_h = C_h$ for $1 \leq h \leq k$, i.e., the first k candidates are trying to bribe voters.

Let the bribers be players in the game. The strategy set of a briber is the set of possible smart contracts he/she can make with voters, i.e., every strategy of a briber b_h is an allocation of the budget b_h among all the voters, which can be represented as an m-vector $(b_h^1, b_h^2, \cdots, b_h^m)$ where b_h^j is the price the briber offers to voter V_j and $\sum_j b_h^j \leq b_h$. The goal of each briber, as a player, is to maximize the (expected) number of votes he/she received.

Nash equilibrium in smart contract bribery game A pure Nash equilibrium for the smart contract bribery game, if it exists, is a solution where every briber B_h specifies some strategy $(b_h^1, b_h^2, \cdots, b_h^m)$ such that if B_h changes his/her strategy unilaterally to some $(\bar{b}_h^1, \bar{b}_h^2, \cdots, \bar{b}_h^m)$, the expected number of votes he/she can get will not increase.

3 The Smart Contract Bribery Game

If there is only one briber, then obviously the briber is able to increase the number of his/her votes if his/her bribing budget is at least as large as the cheapest bribing price of some voter who votes for another candidate. When there are multiple bribers, things become much more complicated. Considering an arbitrary briber, say, B_1, can he/she really benefit from bribery in the presence of other bribers? Of course the answer is no if there exists another briber with an infinite or sufficiently larger budget, who is able to bribe every voter with a price larger than b_1 and B_1 will get no votes at all. If, however, B_1 is more powerful, say $b_1 \geq b_i$ for every $2 \leq i \leq k$, is it possible for B_1 to get additional votes? Unfortunately, this may not necessarily be the case and is highly dependent on the strategies of other bribers. In this section, we focus on Nash equilibrium in the smart contract bribery game. We consider the following problem: In a Nash equilibrium, how many votes can B_1 get when competing against bribers who are weaker than him/her? Furthermore, can B_1 get more votes than he/she gets in the absence of bribery in the system?

Theorem 1. *There may exist a pure Nash equilibrium for the smart contract bribery game where the briber B_1 can get at most $\lfloor 1/\epsilon \rfloor$ votes even if $b_1 \geq 1/\epsilon \cdot b_i$ for every $2 \leq i \leq k$, where $\epsilon \in (0, 1)$ is an arbitrary number.*

We remark that a pure Nash equilibrium may not always exist.

Proof. Consider the following smart contract bribery game in which there are $m = k - 1 + \lfloor 1/\epsilon \rfloor$ voters and exactly k candidates (i.e., $\mathcal{C} = \mathcal{B}$). Let $p_j = 1$ for $1 \leq j \leq k-1$, $p_j = 1/\epsilon$ for $k \leq j \leq m$. Let b_1 be an arbitrary integer larger than $1/\epsilon$, and $b_i = \epsilon b_1$ for every $2 \leq i \leq k$.

Consider the following feasible solution: each briber B_i, $2 \leq i \leq k$, bribes V_{i-1} at the price of ϵb_1. The briber B_1 then bribes V_k to V_m, each at the price of ϵb_1.

It is easy to verify that B_1 gets $\lfloor 1/\epsilon \rfloor$ votes. It suffices to argue that the feasible solution above is a Nash equilibrium. First, we claim that every briber B_i, $2 \leq i \leq k$, will not deviate from the current solution. Note that if B_i aims to bribe some other voter instead of V_{i-1}, then he/she needs to pay at least ϵb_1, for otherwise that voter will simply ignore his/her offer. Therefore, B_i has to take away all the money ϵb_1 from V_{i-1} and bribes some V_h for $h \neq i-1$. However, since V_h already receives ϵb_1 amount of money from another briber, thus in expectation B_i only gets $1/2$ votes, which is worse than the current solution. Hence, B_i will not unilaterally change his/her strategy. Next, we claim that B_1 will not deviate from the current solution. Note that currently B_1 gets one vote at the cost of ϵb_1. If he/she aims at getting votes from any V_h, $1 \leq h \leq k-1$, he/she has two choices. Either he/she pays the price of ϵb_1 and gets $1/2$ votes in expectation, or he/she pays a price strictly larger than ϵb_1 and gets one vote. In both cases, B_1 will lose one vote from the set of voters in $\{V_h : k \leq h \leq m\}$ and get at most one vote from the set of voters $\{V_h : 1 \leq h \leq k-1\}$. □

Note that k is a parameter that can be significantly larger than $1/\epsilon$, Theorem 1 thus implies that a briber may only get a small number of votes even if the bribing budget of any other briber is at most ϵ fraction of his/her budget.

It is worth mentioning that in the proof of Theorem 1 we do not specify which candidate does a voter votes in the absence of bribery. We may assume that without bribery V_h, $1 \leq h \leq k-1$, all vote for B_1, while V_h, $k \leq h \leq m$, all vote for B_2. Therefore, B_1 actually loses an arbitrary amount of votes when bribery happens. More precisely, we have the following corollary.

Corollary 1. *In a smart contract bribery game, a briber may lose an arbitrary number of votes even if he/she is only competing against other bribers whose budget is significantly smaller.*

Theorem 1 implies that the worst Nash equilibrium for a briber can be very bad. However, what if a briber is lucky and ends up in a Nash equilibrium which is the best for him/her? In this case, can the briber win significantly more votes with a very small budget? Unfortunately, even in the best Nash equilibrium, the fraction of the votes a briber can win may not exceed the portion of the bribing budget he/she owns by $O(1)$ times, as is implied by the following theorem.

Theorem 2. *Let $\epsilon < 1/3$ be an arbitrary small constant and suppose $b_i \geq \epsilon b_1$ for $2 \leq i \leq k$. In any Nash equilibrium, B_1 gets at most $1/\epsilon$ votes or a $\frac{4(1+2\epsilon)b_1}{4(1+2\epsilon)b_1 + \sum_{i=2}^{k} b_i}$ fraction of the votes, whichever is larger.*

Proof. Consider an arbitrary Nash equilibrium. If B_1 only gets $1/\epsilon$ votes in expectation, then the theorem is proved. From now on we assume that B_1 receives more than $1/\epsilon$ votes in expectation. In this case, B_1 must have paid less than $b_1\epsilon$ to some voter, say, V_j, who votes for him/her with a positive probability. Since $b_i \geq \epsilon b_1$, the briber B_i must have received a positive number of votes, for otherwise this briber can devote all the budget to V_j and gets one vote, contradicting the fact that the solution is a Nash equilibrium.

Let $\phi_i > 0$ be the expected number of votes received by each briber B_i. We make the following two assumptions.

- each B_i pays out a total price of exactly b_i to voters;
- if B_i gets 0 vote from a voter in expectation, B_i pays 0 to this voter.

The two assumptions are without loss of generality since each B_i gets a positive number of votes from at least one voter, and we can simply let B_i pays all the remaining money in his/her budget to this voter if he/she does not use up the budget. By doing so, B_i cannot get fewer votes. The fact that the original solution is a Nash equilibrium ensures that B_i will not get more votes. Thus, the modified solution is still a Nash equilibrium.

We define the average cost per vote for B_i as $a_i = b_i/\phi_i$. Let S_j be the set of bribers who offers the same highest price for V_j, then every briber $B_i \in S_j$ gets in expectation $1/|S_j|$ votes from V_j. For simplicity we remove all the voters where $S_j = \emptyset$ from now on. We define $x_{ij} \in \{0, 1\}$ as an indicating variable such that $x_{ij} = 1$ if $B_i \in S_j$ and $x_{ij} = 0$ otherwise. Recall that a briber B_i pays b_i^j to V_j, thus we have

$$\sum_{j=1}^{m} x_{ij}/|S_j| = \phi_i, \qquad \forall i \tag{1a}$$

$$\sum_{j=1}^{m} b_i^j x_{ij} = b_i, \qquad \forall i \tag{1b}$$

There are two possibilities with respect to a_1. If $a_1 \geq \epsilon b_1$, then $\phi_1 \leq 1/\epsilon$, which means B_1 gets at most $1/\epsilon$ votes and Theorem 2 is proved. Otherwise $a_1 < \epsilon b_1$ and there are two possibilities.

Case 1. $|\{j : 0 < b_1^j < (1 + 2\epsilon)a_1\}| \leq 1$. Note that a_1 is the average cost. We claim that $\phi_1 < 1/\epsilon$. Otherwise $\sum_{j=1}^{m} x_{1j} \geq 1/\epsilon$ and it follows that $\sum_{j=1}^{m} b_i^j x_{1j} \geq (1+2\epsilon)a_1(\sum_{j=1}^{m} x_{1j} - 1) = (1+2\epsilon)b_1 - (1+2\epsilon)a_1 > b_1$, where the last inequality follows from the fact that $b_1 \geq (1 + 2\epsilon)(1/\epsilon - 1)a_1 = (1 + 1/\epsilon - 2\epsilon)a_1$, whereas $2\epsilon b_1 > (1 + 2\epsilon)a_1$. This, however, is a contradiction to Eq (1b). Therefore, B_1 gets in expectation at most $1/\epsilon$ votes and Theorem 2 is proved.

Case 2. $|\{j : 0 < b_1^j < (1 + 2\epsilon)a_1\}| \geq 2$. In this case, we have the following lemma.

Lemma 1. *If* $|\{j : b_1^j < (1 + 2\epsilon)a_1\}| \geq 2$*, then for any* $2 \leq i \leq k$*,* $a_1 \geq \frac{a_i}{4(1+2\epsilon)}$.

Proof (Proof of Lemma 1). Towards the proof, we need the following claims.

Claim. For every i, there exists some set of voters Γ_i such that $\sum_{j\in\Gamma_i} x_{ij}/|S_j| \leq 1$ and $\sum_{j\in\Gamma_i} b_i^j x_{ij} \geq a_i/2$.

To see the claim, we suppose on the contrary that for every set of voters Γ_i satisfying that $\sum_{j\in\Gamma_i} x_{ij}/|S_j| \leq 1$, it holds that $\sum_{j\in\Gamma_i} b_i^j x_{ij} < a_i/2$. We list all the variables $x_{i1}, x_{i2}, \cdots, x_{im}$ and divide them into q subsets where the h-th subset consists of $x_{i,\ell_{h-1}}, x_{i,\ell_{h-1}+1}, \cdots, x_{i,\ell_h-1}$ for $1 = \ell_0 < \ell_1 < \cdots < \ell_q = m+1$, such that the followings hold for every h:

$$\frac{x_{i,\ell_{h-1}}}{|S_{\ell_{h-1}}|} + \frac{x_{i,\ell_{h-1}+1}}{|S_{\ell_{h-1}+1}|} + \cdots + \frac{x_{i,\ell_h-1}}{|S_{\ell_h-1}|} \leq 1 \tag{2a}$$

$$\frac{x_{i,\ell_{h-1}}}{|S_{\ell_{h-1}}|} + \frac{x_{i,\ell_{h-1}+1}}{|S_{\ell_{h-1}+1}|} + \cdots + \frac{x_{i,\ell_h-1}}{|S_{\ell_h-1}|} + \frac{x_{i,\ell_h}}{|S_{\ell_h}|} > 1 \tag{2b}$$

By Eq (2a) we have

$$\sum_{s=\ell_{h-1}}^{\ell_h-1} b_i^s x_{is} < a_i/2.$$

Taking the summation over $1 \leq h \leq q$, we have

$$\sum_{s=\ell_{h-1}}^{\ell_h-1} b_i^s x_{is} < a_i q/2.$$

We show in the following that $q \leq 2\phi_i$, whereas

$$\sum_{h=1}^{q} \sum_{s=\ell_{h-1}}^{\ell_h-1} b_i^s x_{is} < a_i q/2 \leq a_i \phi_i = b_i,$$

contradicting Eq (1b) and the claim is proved. To see $q \leq 2\phi_1$, we can view each $x_{ij}/|S_j|$ as an item of size $x_{ij}/|S_j|$. We pack these items into bins of size 1 one by one using the *Next-fit* algorithm in Bin packing [30], i.e., as long as the item fits in the same bin as the previous item, put it there; otherwise, open a new bin and put it in there. It is easy to see that the Next-fit algorithm returns a solution using q bins with the h-th bin containing exactly $x_{i,\ell_{h-1}}/|S_{\ell_{h-1}}|$ to $x_{i,\ell_h-1}/|S_{\ell_h-1}|$. Note that $\phi_i = \sum_{j=1}^m x_{ij}/|S_j|$ is exactly the total size of all items. It is a classical result [30] that the Next-fit algorithm for bin packing returns a solution that uses the number of bins at most twice the total item size (to see this, simply observe that any two consecutive bins have a total size larger than 1), hence $q \leq 2\phi_i$.

We are able to prove Lemma 1 now using the above claim. Suppose on the contrary that for some i it holds that $a_1 < a_i/(4+8\epsilon)$. According to the claim, there exists some Γ_i such that $\sum_j \in \Gamma_i x_{ij}/|S_j| \leq 1$ and $\sum_{j\in\Gamma_i} b_i^j x_{ij} \geq a_i/2 > 2(1+2\epsilon)a_1$. Hence, the briber B_i pays in total more than $2(1+2\epsilon)a_1$ and only receive in expectation 1 vote. As $|\{j : 0 < b_1^j < (1+2\epsilon)a_1\}| \geq 2$, there exist at least two voters V_{j_1} and V_{j_2} to whom B_1 pays less than $(1+2\epsilon)a_1$. Since B_1 have received a positive number of votes from each of them (otherwise B_1 would have

paid 0), V_{j_1} and V_{j_2} receive offers from bribers with a price less than $(1+2\epsilon)a_1$. Hence, if B_i changes his/her solution unilaterally by paying $(1+2\epsilon)a_1$ to V_{j_1} and V_{j_2}, and meanwhile 0 to voters in Γ_i, he/she gets 2 votes instead, contracting the fact that the solution is a Nash equilibrium. Thus, Lemma 1 is true. □

By Lemma 1, we know that in Case 2 every briber B_i gets at least $\frac{b_i}{4(1+2\epsilon)a_1}$ votes. Therefore, B_1 can get at most $\frac{4(1+2\epsilon)b_1}{4(1+2\epsilon)b_1+\sum_{i=2}^{k} b_i}$ fraction of the total votes. □

Theorem 2 implies that, even if a briber is very lucky and ends up in a Nash equilibrium which is the best for him/her, he/she cannot get more than $\frac{4(1+2\epsilon)b_1}{4(1+2\epsilon)b_1+\sum_{i=2}^{k} b_i}$ fraction of the total votes if there are significantly many voters (larger than $1/\epsilon$ which is a constant). By taking $\frac{b_1}{\sum_{i=1}^{k} b_i} = 1/5$, this fractional value becomes $1/2 + O(\epsilon)$, therefore we have the following corollary.

Corollary 2. *Even in a best Nash equilibrium, a briber needs to control more than 20% of the total bribing budgets in order to get more than 50% of the votes.*

4 Further Discussion

We have shown that, although smart contracts can be used to carry out bribery in a blockchain system, it is, however, much more difficult for a briber to do so than in an ordinary real-world election. The major challenge comes from the fact that a voter is free to establish multiple smart contracts with different bribers and can strategically pick the best one.

A natural question is whether a briber can prevent a bribee from establishing smart contracts with other bribers. One potential approach is to introduce a penalty for a bribee if he/she fails to fulfill the smart contract. Indeed, a recent paper by Abhiram and Christopher [19] presents a pseudocode for such kind of smart contracts. It is questionable whether such smart contracts can change our results substantially. Obviously, if the briber can charge an infinite amount of penalty, then surely the bribee has no choice but to follow the smart contract. However, this is usually unreasonable. A penalty is usually achieved via a deposit from the bribee to the briber, a sufficiently high penalty may exceed the wallet balance of a voter, which means the briber is losing these potential bribees. More critically, the decision whether a smart contract is fulfilled or not is also achieved through consensus. Once the bribee pays a high deposit, even if he/she fulfills the smart contract, the briber may also bribe others to alter the decision and take away the deposit. Hence, even if penalty may be introduced, it should be reasonably low. A low penalty, however, only prevents a voter from making smart contracts with a lot of bribers. It does not prevent a voter from making smart contracts with only a few bribers, which is already enough to yield a non-cooperative game among bribers and our results readily apply.

5 Related Work

In this section, we briefly review related works.

Smart contract systems. Ethereum is by far the most popular smart contract system [5] and many works have been done to detect potential vulnerabilities in smart contracts, see, e.g., [22]. Although game theory has been extensively used to analyze mining activities [12, 28], users' behavior in a smart contract system is not well understood.

Bribery in elections. There are various researches studying the bribery issue in elections. Faliszewski et al. [14] gave the first systematic characterization on the complexity of the bribery problem where the briber can pay a fixed, but voter-dependent, price to arbitrarily manipulate the preference list of a bribed voter. Different bribery models were addressed subsequently in, e.g., [10, 13, 15, 7, 2, 17]. We refer the readers to [16] for a nice survey on this topic and the references therein.

6 Conclusion and Future Work

Bribery is an important issue in real-world elections. Recent studies have shown that smart contracts can be utilized to conduct bribery in a blockchain system; and it is crucial to understand how smart contract based bribery can influence the whole blockchain system. In this paper, we make the first improvement towards this direction. We cast the bribery problem in a blockchain system as an election and leverage the research in voting systems. We observe that, bribery via smart contracts in a blockchain system is likely to end up in a game situation where different bribers compete with each other in bribing users. We model this problem as a smart contract bribery game and study the behavior of bribers under Nash equilibrium. Interestingly, we show that in *any* Nash equilibrium, a briber cannot win the majority of the votes unless he/she controls more than 20% of the total bribing budgets. Therefore, the phenomenon of "anarchy" in game theory actually helps in discouraging people from carrying out bribery in a blockchain system.

There are several interesting open problems along this line of research. In this paper, we assume every voter has the same weight, i.e., each voter can only cast one vote. However, it is common that voters do have weights. It is not clear whether a constant threshold like 20% also exists when voters/users have weights. Another important problem is to study how to protect the blockchain system through other methods, particularly by deploying resources. It is true that the 20% threshold can discourage people from bribing, but it does not fully defend the system from bribery, especially when some briber owns a large amount of cryptocurrencies. There are several works in the research of voting systems which study the problem of protecting an election by awarding honesty or punishing bribery [37]. It is not clear how to implement a similar scheme in a blockchain system.

References

1. Bachrach, Y., Porat, E.: Path disruption games. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 1123–1130. International Foundation for Autonomous Agents and Multiagent Systems (2010)
2. Bredereck, R., Chen, J., Faliszewski, P., Nichterlein, A., Niedermeier, R.: Prices matter for the parameterized complexity of shift bribery. Information and Computation 251, 140–164 (2016)
3. Bredereck, R., Faliszewski, P., Niedermeier, R., Talmon, N.: Complexity of shift bribery in committee elections. In: AAAI. pp. 2452–2458 (2016)
4. Bredereck, R., Faliszewski, P., Niedermeier, R., Talmon, N.: Large-scale election campaigns: Combinatorial shift bribery. Journal of Artificial Intelligence Research 55, 603–652 (2016)
5. Buterin, V.: A next-generation smart contract and decentralized application platform. white paper (2014)
6. Clark, D.J., Riis, C.: Allocation efficiency in a competitive bribery game. Journal of Economic Behavior & Organization 42(1), 109–124 (2000)
7. Dey, P., Misra, N., Narahari, Y.: Frugal bribery in voting. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 2466–2472. AAAI Press (2016)
8. Dorn, B., Krüger, D.: On the hardness of bribery variants in voting with cp-nets. Annals of Mathematics and Artificial Intelligence 77(3-4), 251–279 (2016)
9. Dorn, B., Krüger, D., Scharpfenecker, P.: Often harder than in the constructive case: destructive bribery in cp-nets. In: International Conference on Web and Internet Economics. pp. 314–327. Springer (2015)
10. Elkind, E., Faliszewski, P., Slinko, A.: Swap bribery. In: International Symposium on Algorithmic Game Theory. pp. 299–310. Springer (2009)
11. Erdélyi, G., Reger, C., Yang, Y.: The complexity of bribery and control in group identification. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. pp. 1142–1150. International Foundation for Autonomous Agents and Multiagent Systems (2017)
12. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: International conference on financial cryptography and data security. pp. 436–454. Springer (2014)
13. Faliszewski, P.: Nonuniform bribery. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3. pp. 1569–1572. International Foundation for Autonomous Agents and Multiagent Systems (2008)
14. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.A.: How hard is bribery in elections? Journal of Artificial Intelligence Research 35, 485–532 (2009)
15. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.A., Rothe, J.: Llull and copeland voting computationally resist bribery and constructive control. Journal of Artificial Intelligence Research 35, 275–341 (2009)
16. Faliszewski, P., Rothe, J.: Control and Bribery in Voting. Cambridge University Press (2016)
17. Kaczmarczyk, A., Faliszewski, P.: Algorithms for destructive shift bribery. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. pp. 305–313. International Foundation for Autonomous Agents and Multiagent Systems (2016)

18. Knop, D., Koutecký, M., Mnich, M.: Voting and bribing in single-exponential time. In: LIPIcs-Leibniz International Proceedings in Informatics. vol. 66. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
19. Kothapalli, A., Cordi, C.: A bribery framework using smartcontracts (2017)
20. Lewenberg, Y., Bachrach, Y., Sompolinsky, Y., Zohar, A., Rosenschein, J.S.: Bitcoin mining pools: A cooperative game theoretic analysis. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 919–927. International Foundation for Autonomous Agents and Multiagent Systems (2015)
21. Lien, D.H.D.: A note on competitive bribery games. Economics Letters 22(4), 337–341 (1986)
22. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 254–269. ACM (2016)
23. Mattei, N., Pini, M.S., Venable, K.B., Rossi, F.: Bribery in voting over combinatorial domains is easy. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3. pp. 1407–1408. International Foundation for Autonomous Agents and Multiagent Systems (2012)
24. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
25. Osborne, M.J., Rubinstein, A.: A course in game theory. MIT press (1994)
26. Pini, M.S., Rossi, F., Venable, K.B.: Bribery in voting with soft constraints. In: AAAI (2013)
27. Rey, A., Rothe, J.: Bribery in path-disruption games. In: International Conference on Algorithmic Decision Theory. pp. 247–261. Springer (2011)
28. Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in bitcoin. arXiv preprint arXiv:1507.06183 (2015)
29. Szabo, N.: Formalizing and securing relationships on public networks. First Monday 2(9) (1997)
30. Vazirani, V.V.: Approximation algorithms. Springer Science & Business Media (2013)
31. Vukolić, M.: The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In: International Workshop on Open Problems in Network Security. pp. 112–125. Springer (2015)
32. Xu, L., Chen, L., Gao, Z., Lu, Y., Shi, W.: Coc: Secure supply chain management system based on public ledger. In: Computer Communication and Networks (ICCCN), 2017 26th International Conference on. pp. 1–6. IEEE (2017)
33. Xu, L., Chen, L., Shah, N., Gao, Z., Lu, Y., Shi, W.: Dl-bac: Distributed ledger based access control for web applications. In: Proceedings of the 26th International Conference on World Wide Web Companion. pp. 1445–1450. International World Wide Web Conferences Steering Committee (2017)
34. Xu, L., Shah, N., Chen, L., Diallo, N., Gao, Z., Lu, Y., Shi, W.: Enabling the sharing economy: Privacy respecting contract based on public blockchain. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts. pp. 15–21. ACM (2017)
35. Yang, Y., Shrestha, Y.R., Guo, J.: How hard is bribery in party based elections? In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 1725–1726. International Foundation for Autonomous Agents and Multiagent Systems (2015)
36. Yang, Y., Shrestha, Y.R., Guo, J.: How hard is bribery with distance restrictions? In: ECAI. pp. 363–371 (2016)

37. Yin, Y., Vorobeychik, Y., An, B., Hazon, N.: Optimally protecting elections. In: IJCAI. pp. 538–545 (2016)

Verifiable Sealed-Bid Auction on the Ethereum Blockchain

Hisham S. Galal and Amr M. Youssef

Concordia Institute for Information Systems Engineering,
Concordia University, Montréal, Quebéc, Canada

Abstract. The success of the Ethereum blockchain as a decentralized application platform with a distributed consensus protocol has made many organizations start to invest into running their business on top of it. Technically, the most impressive feature behind Ethereum's success is its support for a Turing complete language. On the other hand, the inherent transparency and, consequently, the lack of privacy poses a great challenge for many financial applications. In this paper, we tackle this challenge and present a smart contract for a verifiable sealed-bid auction on the Ethereum blockchain. In a nutshell, initially, the bidders submit homomorphic commitments to their sealed-bids on the contract. Subsequently, they reveal their commitments secretly to the auctioneer via a public key encryption scheme. Then, according to the auction rules, the auctioneer determines and claims the winner of the auction. Finally, we utilize interactive zero-knowledge proof protocols between the smart contract and the auctioneer to verify the correctness of such a claim. The underlying protocol of the proposed smart contract is partially privacy-preserving. To be precise, no information about the losing bids is leaked to the bidders. We provide an analysis of the proposed protocol and the smart contract design, in addition to the estimated gas costs associated with the different transactions.

Keywords: Ethereum, Smart Contract, Sealed-Bid Auction.

1 Introduction

Online auctions have played an important role in the world economy by transferring trillions of dollars in exchange for goods and services in the recent decades. An auction is a platform for sellers to advertise the sale of arbitrary assets where buyers place competitive bids as the highest prices they are willing to pay. Practically, auctions promote many economic advantages for the efficient trade of goods and services. Traditionally, there are four main types of auctions [7]:

1. First-price sealed-bid auctions (FPSBA). Bidders submit their bids in sealed envelopes and hand them to the auctioneer. Subsequently, the auctioneer opens the envelopes to determine the bidder with the highest bid.
2. Second-price sealed-bid auctions (Vickrey auctions). It is similar to FPSBA with the exception that the winner pays the second highest bid instead.

3. Open ascending-bid auctions (English auctions). Bidders increasingly submit higher bids and stop bidding when they are not willing to pay more than the current highest bid.
4. Open descending-bid auctions (Dutch auctions). Auctioneer initially sets a high price, which is gradually decreased until a bidder decides to pay at the current price.

Arguably, the main advantage behind the sealed-bid auctions lies in the fact that no bidder learns any information about the other bids. Hence, the bidders are encouraged to bid according to their monetary valuation of the asset. However, a collusion between the auctioneer and a malicious bidder can break this advantage. In other words, there is a conflict between preserving the privacy of the bids and trusting the auctioneer to individually determine the winner. Hence, in online sealed-bid auctions, cryptographic protocols can be utilized to accomplish the publicly verifiable correctness without sacrificing the privacy of the bids.

According to a recent Reuters report [10], as part of the efforts to improve the transparency in government transactions, the Ukraine's justice ministry carried out trial auctions on top of the blockchain. The main goal is to make the auction system more transparent and secure such that the information is accessible to everyone to check if there is any manipulation or corruption.

Recently, cryptocurrencies have gained high popularity as evidenced by the surge in Bitcoin exchange rate. The foundation of cryptocurrencies is based on a decentralized public ledger on a peer-to-peer network that maintains the history of all transactions in an append-only fashion. Peers agree on the state of the ledger through an incentive-based consensus protocol. Additionally, cryptocurrencies also use cryptography to secure transactions as well as to control the creation of new currency units. Furthermore, many cryptocurrencies blockchains go beyond the simple means of payments. In fact, they provide a support for building and executing contracts on top of them. Simply, a smart contract is a piece of code that is stored and run on the blockchain. The smart contract resides passive until its execution is triggered by transactions. With the help of the consensus protocol, the contract is also guaranteed to be executed as its code dictates.

The Ethereum blockchain [17] presumably provides the highest support for smart contracts creation. Smart contracts are executed by a simple stack-based Turing complete 256-bit virtual machine known as the Ethereum Virtual Machine (EVM). Solidity is the common scripting language for writing smart contracts with a growing community. Ether represents the unit of currency in Ethereum and there are two types of accounts: externally owned accounts and contract accounts. An externally owned account is typically associated with a user, it consists of a unique public-private key pair. On the other hand, a contract account is controlled by the contract instead of a single private key. Transactions are created and signed by externally owned accounts. The receiver of the transaction can be an externally owned account or a contract account. In the former case, the transaction's purpose is to transfer ethers between users. Whereas in the latter case, the transaction triggers the execution of a function on the smart

contract. Transactions also include a gas limit and a gas price; the amount of gas consumed to execute the transaction is converted into ethers using the gas price. These ethers are charged to the sender's account as transaction fees.

The Ethereum project has been planned in four locksteps [16]: Frontier, Homestead, Metropolis, and Serenity. Each update brings a set of approved Ethereum Improvement Proposals (EIP). Recently, the Ethereum blockchain has been upgraded to the first phase of Metropolis which is named Byzantium. The fork has been announced by the Ethereum team at the block number 4,370,000 [14]. Byzantium includes EIP-196 and EIP-197 to efficiently perform elliptic curve point addition and scalar multiplication operations on *alt_bn128* curve [15]. Simply, they are precompiled contracts with special addresses that are intercepted by the client software which provide efficient native implementation, rather than the inefficient EVM implementation, for elliptic curve operations. These proposals prepare Ethereum for untraceable transactions by integrating ZK-Snarks, a cryptographic innovation developed in collaboration with the anonymity-centric cryptocurrency Zcash [12].

Despite the flexibility and power of the smart contracts, the present form of the blockchain technologies lacks transactional privacy. Typically, every sequence of actions executed in the smart contract is propagated across the network and ends up being recorded on the blockchain. As a result, the lack of privacy is considered a major challenge towards the adoption of smart contracts as alternatives to many financial applications. Many individuals are not willing to reveal their financial transactions to the public. In this paper, we tackle this challenge and present an auction smart contract that utilizes a set of cryptographic primitives to guarantee the following attributes:

1. Bid privacy. All bidders cannot know the bids submitted by the others before committing to their own. This property is also guaranteed even in a collusion with a malicious auctioneer.
2. Posterior privacy. Given a semi-honest auctioneer, all committed bids are maintained private from the bidders and public users.
3. Non-repudiability. Once the bid interval is closed, bidders cannot change or deny the commitments to their sealed-bid.
4. Public verifiable correctness. The auction contract verifies the correctness of the auctioneer's work to determine the auctioneer winner.
5. Financial fairness. Bidders or auctioneer may attempt to deviate from the protocol and prematurely abort to affect the behavior of the auction protocol. The aborting parties are financially penalized while honest parties are refunded after a specific timeout.
6. Non-Interactivity. Bidders do not participate in complex interactions with the underlying protocol of the auction contract. In fact, no extra communications between the bidders and the auction contract are required aside from the submission of the bid commitments and the associated opening values.

We have also made our implementation prototype available on Github [1] for researchers and community to review it.

[1] `https://github.com/HSG88/AuctionContract`

The rest of this paper is organized as follows. Section 2 provides a review of state-of-the-art research on auction solutions on the blockchain. The cryptographic primitives and the protocol for comparing the bids and verifying the correctness of the auction winner are presented in Section 3. In Section 4, we provide an analysis of the auction contract design and the estimated gas cost of the relevant transactions. Finally, we present our conclusions and future work in Section 5.

2 Related Work

Many of the previous research have focused on combining cryptocurrencies with secure multiparty computation protocols (MPC) and/or zero-knowledge proofs (ZKP). Typically, the cryptocurrency is used to incentive fairness and correctness, and avoid deviations from the MPC or ZKP protocol [1,2,8,9]. Initially, each participant deposits an amount of cryptocurrency in a smart contract. These funds are reserved while the protocol is still running. Subsequently, once the protocol reaches a final state after an arbitrary timeout, the deposits get refunded only to the honest players. This in effect encourages parties to strictly follow the protocols to avoid the financial penalty.

In [6], the authors presented Hawk, a framework for creating Ethereum smart contract that does not store financial transactions in the clear on the blockchain. One can easily write a Hawk program without having to implement any cryptography. The associated compiler utilizes different cryptographic primitives such as ZKP to automatically generate privacy-preserving smart contracts. A Hawk program contains public and private parts. The public part consists of the logic that does not deal with the data or the currency. Conversely, the private part is responsible for hiding the information about data and input currency units. The compiler translates the Hawk program into three pieces that define the cryptographic protocol between users, manager, and the blockchain nodes. The security of a Hawk program is guaranteed to satisfy *on-chain privacy* that protects the flow of money and data from the public view, and *contractual security* that protects the parties in the agreement of the contract from each other. Up to our knowledge, the Hawk framework has not been released yet on the project homepage `http://oblivm.com/hawk/download.html`.

The authors in [3] presented *Strain*, a protocol to implement sealed-bid auctions on top of blockchains that protects the bid privacy against fully-malicious parties. To achieve efficiency and low latency cost, the authors avoided the use of highly interactive MPC primitives such as garbled circuits. Instead, they designed a two-party comparison mechanism executed between any pair of bidders in parallel. The outcome of the comparison is broadcasted to all bidders such that each one can verify it using ZKP. An additional ZKP protocol is used to verify that the comparisons only involved the committed bids. Moreover, to achieve fairness against prematurely aborting malicious parties, the protocol uses a reversible commitment scheme such that a group of bidders can jointly open the

bid commitment. The authors mentioned that the proposed protocol leaks the order of bids similar to Order Preserving Encryption (OPE) schemes.

In [13], the author proposed *Raziel*, a system that combines MPC and ZKP to guarantee the privacy, correctness and verifiability of smart contracts. The associated proofs of the smart contracts can effectively prove the functional correctness of a computation, besides to additional properties such as termination, security, pre-conditions and post-conditions. Furthermore, the author presented how a smart contract owner can prove its validity to third parties without revealing any information about the source code by using Zero-Knowledge Proofs to create Proof-Carrying Code certificates. Moreover, the author also proposed an incentive-based scheme for miners to genereate preprocessed data of MPC.

3 Preliminaries

In this section, we briefly explain the cryptographic primitives that are utilized in the design of our proposed protocol:

1. Homomorphic commitment scheme that supports the addition operation on the underlying values
2. Zero-knowledge proof of interval membership $x \in [0, B]$.

3.1 Homomorphic Commitment Scheme

Our protocol makes an extensive use of Pedersen commitment scheme [11]. Let G and H be fixed public generators of the elliptic curve *alt_bn128* which is supported in EIP-196 and EIP-197 with the group order q [15]. The value of H is chosen such that neither the bidders nor the auctioneer know its discrete log. To commit a bid $x \in Z_q$, the bidder chooses a random $r \in Z_q$, then computes the commitment as $C = xG + rH$. Later, to open the commitment C, the bidder simply reveals the values of x and r. The Pedersen commitment scheme also possesses the homomorphic addition property on the underlying committed values by simply computing the point addition operation on the commitments. In other words, given two commitments $C_1 = x_1G+r_1H$ and $C_2 = x_2G+r_2H$, then $C_1+C_2 = (x_1+x_2)G+(r_1+r_2)H$ which is essentially the outcome commitment to $x_1 + x_2$.

3.2 Zero-Knowledge Proof of Interval Membership

We adapt the interval membership ZKP protocol proposed in [4]. Given an arbitrary number x which belongs to an interval $[0, B)$, the prover is able to convince the verifier that $x \in [-B, 2B)$. Since the financial values of bids cannot be negative numbers, the proved interval membership becomes $x \in [0, 2B)$. The protocol runs as follows:

1. **Commit.** The prover picks a number $w_1 \in [0, B]$ and sets $w_2 = w_1 - B$. Then, the prover sends the commitments $X = xG + uH, W_1 = w_1G + r_1H$, and $W_2 = w_2G + r_2H$ to the verifier.

2. **Challenge.** The verifier picks a random variable $b \in \{0, 1\}$.
3. **Response.** The prover sends one of the following responses to the verifier based on the value of b:
   - Case $b = 0$, the prover sends w_1, r_1, w_2, and r_2. The verifier checks $|w_1 - w_2| = B$, and the successful openining of the commitments W_1 and W_2.
   - Case $b = 1$, the prover sends $m = x + w_z$ and $n = u + r_z$, where $m \in [0, B)$ and $z \in \{0, 1\}$. The verifier checks $XW_z = (x + w_z)G + (u + r_z)H$.

In this protocol, the probability of cheating is $\frac{1}{2}$ which is non-negligible. However, with multiple k rounds of the protocol, the cheat probability becomes $\frac{1}{2^k}$.

3.3 Proving Claimed Inequality $x_1 > x_2$

Based on the primitives outlined above, we can prove that one bid is greater than another as follows. Suppose that $x_1, x_2 \in Z_q$, where q is a 256-bit prime number representing the order of *alt_bn128* elliptic curve as specified in EIP-197 and EIP-198 [15]. Then it is relatively easy to prove that $x_1 > x_2$ if and only if the following three interval membership hold (i) $x_1 \in [0, \frac{q}{2})$, (ii) $x_2 \in [0, \frac{q}{2})$, and (iii) $\Delta x_{1,2} \in [0, \frac{q}{2})$ where $\Delta x_{1,2} = (x_1 - x_2) \, mod \, q$.

In our work, the auctioneer acts as a prover and the auction contract acts as a verifier. Recall that in the interval membership ZKP, the prover is able to convince the verifier that $x \in [0, 2B)$ given that $x \in [0, B)$. As a result, we set an upper bound $V = \frac{q}{4}$ on the range of possible bids. Additionally, the auctioneer is not allowed to create any commitments for the bids, instead, the auctioneer only uses the commitments submitted by the bidders on the smart contract. The auction contract utilizes the additive homomorphic feature of Pedersen commitment scheme to compute the commitment to the differences between each pair of bids $\Delta X_{i,j} = X_i + (-1)X_j$.

4 Auction Smart Contract

In this section, we illustrate all the interactions between the bidders, the auctioneer, and the auction contract. Although our work applies to both types of sealed-bid auctions, we demonstrate the interactions in the case of FPSBA.

There are five sequential phases from the initial deployment of the auction contract to the collection of the highest bid from the winner given a successful verification of correctness. There are two methods to define phases of a smart contract: time interval and block interval. In time interval, the smart contract checks the time of the mined block (*block.timestamp or now*) which is specified by the block's miner. Ethereum developers discourage this method since it can be easily manipulated by the miners. On the other hand, in block interval, the smart contract loses the notion of time.

4.1 Phase 1: Contract Deployment and Parameters Setup

As shown in Fig. 1, the auctioneer initially deploys the auction contract on the Ethereum blockchain with the following set of parameters:

Create: upon receiving from auctioneer `A` $(T_1, T_2, T_3, T_4, N, F, A_{pk})$:

```
Set state := INIT, bidders := {}, zkpCommits := {}
Set highestBid := 0, winner := 0
Set challengeBlockNumber := 0, challengedBidder := 0
Assert T < T_1 < T_2 < T_3 < T_4
Assert ledger[A] >= F
Set ledger[A] := ledger[A] - F
Set deposit := deposit + F
```

Fig. 1. Pseudocode for the deployment of the auction contract

1. T_1, T_2, T_3, T_4 define the time intervals for the following four phases: commitments of bids, opening the commitments, verification of the winner, and finalizing the auction, respectively.
2. F defines the amount of initial deposit of ethers received from the bidders and the auctioneer to achieve financial fairness against malicious parties.
3. N is the maximum number of bidders.
4. A_{pk} is the auctioneer's public key of an asymmetric encryption scheme.

4.2 Phase 2: Commitment of Bids

This phase starts immediately after the deployment of the auction contract. Each bidder submits a bid commitment using Pedersen commitment scheme along with the initial deposit F in ethers to the function `Bid` as shown in Fig. 2.

Bid: upon receiving from a bidder `B` (com_B):

```
Assert T < T_1
Assert ledger[B] > F
Set ledger[B] := ledger[B] - F
Set deposit := deposit + F
Set bidders[B].Commit := com_B
```

Fig. 2. Pseudocode for the Bid function

Suppose that an arbitrary bidder Bob is known to be very rich and is really interested in winning the auctioned item, i.e., Bob is very likely to be the one who submits the highest bid. Then, a collusion between a malicious bidder Alice and the auctioneer can eliminate Bob's winning chance by abusing the homomorphic property of the Pedersen commitment. The attack can be carried out as follows:

1. Bob submits the commitment $C_B = (xG + rH)$.

2. Subsequently, Alice submits the commitment $C_A = C_B + (G + H)$.
3. Bob reveals (x, r) to the auctioneer.
4. The auctioneer forwards (x, r) to Alice.
5. Alice reveals $(x + 1, r + 1)$.

To avoid this attack, we utilize Chaum-Pedersen non-interactive ZKP [5], which is not shown in Fig. 2. for the sake of simplicity. In this case, the above attack is not applicable because Bob sends commitments to random numbers rather than the actual bid which are subsequently challenged to verify the knowledge of values (x, r). As a result, Alice cannot succeed to imitate Bob's commitment since she will receive different challenges to verify the knowledge of $(x+1, r+1)$.

4.3 Phase 3: Opening the Commitments

Each bidder B_i sends the outcome ciphertext of encrypting (x_i, r_i) by the public key of the auctioneer A_{pk} to the function `Reveal` on the auction contract as shown in Fig. 3.

Reveal: upon receiving from a bidder `B` (*ciphertext*):
`Assert` $T_1 < T < T_2$
`Assert B` $\in$ `bidders`
`Set bidders[B].Ciphertext := ciphertext`

Fig. 3. Pseudocode for the Reveal function

The ciphertexts are stored on the auction contract instead of being sent directly to the auctioneer in order to avoid the following attack scenario. Suppose a malicious auctioneer pretends that an arbitrary bidder Bob has not revealed the opening values of the associated commitment. In this case, Bob has no chance of denying this false claim. However, if the ciphertexts are to be stored on the auction contract, then their mere existence successfully prevents this attack.

We have also taken into our account the possibility of the following attack as well. Suppose a malicious auctioneer intends to penalize an arbitrary bidder Bob by claiming that the decryption outcome of Bob's ciphertext CT_B does not successfully open Bob's commitment C_B. We prevent this attack by requiring the auctioneer to verify the opening correctness of the commitments once they are submitted by the bidders. In the case of unsuccessful opening, the auctioneer declares on the auction contract that the ciphertext associated with the bidder B is invalid. The honest bidder can deny this claim by revealing (x_B, r_B) to the auction contract. Subsequently, the auction contract encrypts the revealed values by the public key A_{pk}. If the outcome ciphertext is found to be equivalent to the previously submitted ciphertext, then the auction contract penalizes the auctioneer and terminates the auction after refunding the bidders. Otherwise, the bidder is penalized and the associated commitment is removed, such that only the valid commitments exist on the auction contract.

To guard against *forward search* attack on the submitted ciphertexts, the parameter r in the opening values is a 256-bit random number that has no restriction on its value compared to the parameter x. Additionally, the opening values are combined to form one message which is passed to the encryption scheme.

4.4 Phase 4: Verification of Comparison Proofs

The auctioneer orders the bids to determine the wining bid x_w, the associated account address B_w and commitment C_w. Then, the auctioneer has to prove that $x_w > x_i$ for all $i \neq w$ and $0 < i < N$. The auction contract has a set of states to impose an order on the functions being invoked by the auctioneer for verification. Initially, the auctioneer calls the function `ClaimWinner` to claim that a winner is found by specifying the account address and opening values of the bid commitment as shown in Fig. 4.

ClaimWinner: upon receiving from auctioneer `A` (B_w, x_w, r_w):

```
Assert state = INIT
Assert T_2 < T < T_3
Assert x_w < V
Assert B_w ∈ bidders
Assert bidders[B_w].commit = Pedersen.Commit(x_w, r_w)
Set winner := B_w
Set highestBid := x_w
Set state := Challenge
```

Fig. 4. Pseudocode for the ClaimWinner function

Recall that the interval membership ZKP has a probability of cheating $\frac{1}{2}$ which is non-negligible; however, this probability can be further reduced to $(\frac{1}{2})^k$ by running the protocol k times. Moreover, in the *challenge* step, the verifier sends to the prover a random value $b \in \{0, 1\}$ which has to be non-predictable. However, smart contracts cannot send data to externally owned accounts, (i.e., the auction contract cannot send a challenge value to the auctioneer). Hence, we utilize a non-interactive interval membership ZKP to prove $x_i \in [0, \frac{q}{2})$ as follows:

1. **Commit:** The auctioneer chooses k-pairs of $(w_{1,j}, w_{2,j})$ where $w_{1,j} \in [-V, V)$ and $w_{2,j} = w_{1,j} - V$ such that $|w_{1,j} - w_{2,j}| = V$ for $1 \leq j \leq k$. Then, the auctioneer invokes the function `ZKPCommit` with the account address of the challenged bidder and the commitments to w_1 and w_2 as shown in Fig. 5.

ZKPCommit: upon receiving from auctioneer `A` $(B_i, commits)$:

```
Assert state = Challenge
Assert T2 < T < T3
Assert Bi ∈ bidders
Set zkpCommits :=commits
Set challengeBidder := Bi
Set challengeBlockNumber := QueryBlockNumber()
Set State := Verify
```

Fig. 5. Pseudocode for the ZKPCommit function

2. **Challenge and Response:**
   - The auctioneer receives a transaction receipt which includes the hash of the block containing the transaction after it has been confirmed. The `ZKPCommit` function has no access to this hash while it is being executed; therefore it stores the current block number in `challengeBlockNumber`.
   - The least significant k-bits of the hash are chosen as the challenge b_j.
   - The auctioneer creates k responses R_j based on the values of b_j.
   - Case $b_j = 0$, then $R_j = \{w_{1,j}, r_{1,j}, w_{2,j}, r_{2,j}\}$.
   - Case $b_j = 1$, then $R_j = \{m_j, n_j, z\}$ where $m_j = x_j + w_{z,j}$, $n_j = u_j + r_{z,j}$ such that $m_j \in [0, V)$ and $z \in \{1, 2\}$.
   - The auctioneer invokes the function `ZKPVerify` with input parameter *responses* which is an array of R_j as shown in Fig. 6.

ZKPVerify: upon receiving from auctioneer `A` $(responses)$

```
Assert State = Verify
Assert T2 < T < T3
Set hash := QueryBlockHash(challengeBlockNumber)
for j ∈ [1,k], Rj ∈ responses, Cj ∈ zkpCommits
    Set bj := Bit(hash,j)
    if bj = 0
        Assert VerifyFirstCase(Cj, Rj)
    else
        Assert VerfiySecondCase(Cj, Rj)
Set bidders[challengeBidder].ValidBid := true
Set state := Challenge
```

Fig. 6. Pseudocode for the ZKPVerify function

As explained in Section 3, three interval membership ZKP are required to prove that $x_w > x_i$. However, since the bid of the winner B_w is revealed, then the number of proofs is reduced to two. In other words, the auctioneer has to prove the interval membership for all bids x_i other than the winning bid and their associated differences Δ_{wi}. The function `ZKPCommit` and `ZKPVerify` contain extra logic to also verify the correctness of $\Delta_{wi} \in [0, \frac{q}{4})$.

4.5 Phase 5: Finalizing the Auction

After the successful verification of correctness, the auctioneer invokes the function `VerifyAll` as shown in Fig. 7 to change the state of the auction contract so that the winner can pay the winning bid.

VerifyAll upon receiving from auctioneer `A` ()

```
Assert state = Challenge
Assert T_2 < T < T_3
For all b ∈ bidders - {winner}
    Assert b.ValidBid = true and b.ValidDelta = true
Set State := ValidWinner
```

Fig. 7. Pseudocode for the VerifyAll function

Subsequently, The winner invokes the function `WinnerPay` to deposit the difference between the winning bid and the initial deposit F as shown in Fig. 8.

WinnerPay upon receiving from a bidder `B` ("*winnerPay*")

```
Assert State = ValidWinner
Assert T_3 < T < T_4
Assert B = winner
Assert ledger[B] > highestBid - F
Set ledger[B] := ledger[B] - highestBid +F
Set deposit := deposit + highestBid - F
Set state := WinnerPaid
```

Fig. 8. Pseudocode for the WinnerPay function

The auction contract guarantees to refund the initial deposit to all honest players after the time T_3 as shown in Fig. 9. In the case of invalid proofs, it penalizes the auctioneer and refunds all bidders. Otherwise, it refunds the losing bidders and the auctioneer as well. It is also clear that the only way for the winner to refund the initial deposit is by invoking WinnerPay function.

Timer

```
if T > T_3 then
     if state ≠ ValidProof then
         refund(F) for all b ∈ bidders
     else
         refund(F) to auctioneer A
         refund(F) for all b ∈ bidders - {winner}
```

Fig. 10. Pseudocode for the Timer function

4.6 Gas Cost

We have created a local private Ethereum blockchain to test our prototype using the *Geth* client version 1.7.2. To support the Byzantium EIP-196 and EIP-197, the *genesis.json* file has to contain the attribute {*"byzantiumBlock": 0*}. Additionally, since Ethereum does not support timer triggered functions, we have implemented a *Withdraw* function that is invoked by an explicit request from the honest players to refund their initial fairness deposit. We have tested the auction contract with ten bidders, and we have set $k = 10$ as the number of multiple rounds to verify interval membership NiZKP which results in a probability of cheat less than 0.001. The upper bound on bid values is up to 250-bit length which is very adequate for financial values. The Pedersen commitment size is 512-bits that represent two points on the elliptic curve. The ciphertext submitted to the `Reveal` function is 1024-bits. Table 1 shows the consumed gas and the equivalent monetary cost in *US* dollars for invoking different functions on the auction contract. As of November 30, 2017, the ether exchange rate is 1 ether = 450$ and the gas price is approximately 20 $Gwei = 20 \times 10^{-9}$ ether. Furthermore, the execution of "heavy" functions in Ethereum is not only costly in dollar terms, but may be even impossible, if the function's gas requirements exceed the block gas limit. The block gas limit at time of writing is 8m gas, whereas the most expensive protocol function consumes 2m gas, which seems feasible

Table 1. Consumed gas cost for different functions of the Auction contract

Function	Gas units	Gas cost (USD)
Deployment	3131261	28.18
Bid	130084	1.17
Reveal	132849	1.19
ClaimWinner	166288	1.49
ZKPCommit	656689	5.91
ZKPVerify	2002490	18.02
VerifyAll	46580	0.42
Withdraw	47112	0.42

5 Conclusion and Future Work

In this paper, we presented a smart contract for a verifiable sealed-bid auction on the Ethereum blockchain. We utilized Pedersen commitment scheme along with ZKP of interval membership to create the underlying protocol. The auction contract maintains the privacy of bids such that bidders do not learn any information about the other bids when they commit. Additionally, the auction

contract also exhibits the public verifiable correctness as it is designed to verify the proofs claimed by the auctioneer to determine the winner. Moreover, no complex interaction is required from the bidders other than submitting and revealing the commitments to their bids. The proposed protocol can be easily modified to support the full privacy of all bids including the winner's bid if there is a desire to receive the payment of winning bid aside from the blockchain. For future work, we will investigate other approaches applicable to the Ethereum blockchain where we can also protect the privacy of bids from all parties including the auctioneer.

References

1. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 443–458. IEEE, 2014.
2. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *International Cryptology Conference*, pages 421–439. Springer, 2014.
3. Erik-Oliver Blass and Florian Kerschbaum. Strain: A secure auction for blockchains. Cryptology ePrint Archive, Report 2017/1044, 2017. https://eprint.iacr.org/2017/1044.
4. Ernest F Brickell, David Chaum, Ivan B Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 156–166. Springer, 1987.
5. David Chaum and Torben P Pedersen. Wallet databases with observers. In *Crypto*, volume 92, pages 89–105. Springer, 1992.
6. Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858. IEEE, 2016.
7. Vijay Krishna. *Auction theory*. Academic press, 2009.
8. Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 418–429. ACM, 2016.
9. Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 406–417. ACM, 2016.
10. Alessandra Prentice Olena Vasina. Ukrainian ministry carries out first blockchain transactions. Reuters Technology News. https://goo.gl/J8X1up.
11. Torben Pedersen and Bent Petersen. Explaining gradually increasing resource commitment to a foreign market. *International business review*, 7(5):483–501, 1998.
12. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.
13. David Cerezo Snchez. Raziel: Private and verifiable smart contracts on blockchains. Cryptology ePrint Archive, Report 2017/878, 2017. https://eprint.iacr.org/2017/878.

14. Ethereum Project Team. Byzantium hf announcement, 2017. https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/.
15. Ethereum Project Team. Ethereum improvement proposals, 2017. https://github.com/ethereum/EIPs.
16. Ethereum Project Team. The ethereum launch process, 2017. https://blog.ethereum.org/2015/03/03/ethereum-launch-process/.
17. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.

Models for Smart Contracts: present and future perspectives

Arthur Breitman
(Tezos Founder)

Abstract. While most legal contracts are bilateral, or involve few parties, many popular smart-contracts involve a large number of participants interacting concurrently.
I'll examine some of the challenges introduced by this model in terms of safety, privacy, and scalability, and discuss potential remedies.

Lightweight Blockchain Logging for Data-Intensive Applications

Yuzhe Tang † Zihao Xing † Cheng Xu ‡ Ju Chen † Jianliang Xu ‡

†*Syracuse University, NY, USA, Email:* {ytang100,zixing,jchen133}@syr.edu

‡*Hong Kong Baptist University, Kowloon Tong, Hong Kong, Email:* {chengxu, xujl}@comp.hkbu.edu.hk

Abstract

With the recent success of cryptocurrency, Blockchain's design opens the door of building trustworthy distributed systems. A common paradigm is to repurpose the Blockchain as an append-only log that logs the application events in time order for subsequent auditing and query verification. While this paradigm reaps the security benefit, it faces technical challenges especially when being used for data-intensive applications.

Instead of treating Blockchain as a time-ordered log, we propose to lay the log-structured merge tree (LSM tree) over the Blockchain for efficient and lightweight logging. Comparing other data structures, the LSM tree is advantageous in supporting efficient writes while enabling random-access reads. In our system design, only a small digest of an LSM tree is persisted in the Blockchain and minimal store operations are carried out by smart contracts. With the implementation in Ethereum/Solidity, we evaluate the proposed logging scheme and demonstrate its performance efficiency and effectiveness in cost saving.

I. Introduction

Recent years witnessed the advent and wide adoption of the first cryptocurrency, BitCoin [1], followed by many others including Ethereum [2], Litecoin [3], Namecoin [4], etc. The initial success of cryptocurrency demonstrates the trustworthiness of Blockchain, the underlying platform of cryptocurrency. The Blockchain supports the storage and processing of cryptocurrency transactions. In abstraction, it is a trust-decentralized network storing transparent state designed with incentives to enable open membership at scale. A line of the latest research and engineering aims at applying the trustworthy design of Blockchain for applications beyond cryptocurrency.

A common paradigm of repurposing Blockchain is to treat the Blockchain as a public append-only log [5], where application-level events are logged into the Blockchain in the order of time, and the log is used later for verification and auditing. While this public-log paradigm reaps the security benefit of Blockchain, it is limited to the applications handling small data (due to high Blockchain storage cost) and tolerating long verification delay (linear scanning the entire chain for verification).

In this work, we tackle the research of repurposing Blockchains for hardening the security of data-intensive applications hosted in a third-party platform (e.g., cloud). A motivating scenario is to secure the cloud-based Internet-of-things (IoT) data storage where the IoT data producers continuously generate an intensive stream of data writes to the third-party cloud storage which serves data consumers through queries. Including Blockchain could enhance the trustworthiness of the third-party cloud storage.

A baseline approach is to log the sequence of data writes in the time order into the Blockchain, in a similar way to log-structured file systems [6]. This approach causes a high read latency (linear to the data size). Another baseline is to digest the latest data snapshot, e.g., using Merkle tree, and place the digest inside the Blockchain. In the presence of dynamic data, the digest scheme usually follows classic B-tree alike data structures [7], [8] that perform "in-place" updates. These schemes incur high write amplification as writing a record involves a read-modify-write sequence on the tree and has $O(logN)$ complexity per write. On Blockchain, this high write amplification causes high cost, as writing a data unit in Blockchain is costly (which involves duplicated writes on miners and expensive proof-of-work alike computation). The problem compounds especially in the write-intensive applications as IoT streams.

To log write-intensive applications using write-expensive Blockchain, we propose to place the log-structured merge tree (LSM tree) [9] over the Blockchain for efficient and lightweight logging. An LSM tree is a write-optimized data structure which supports random-access reads; comparing the above two baselines (append-only log as in log-structured file system and update-in-place structures in database indices), an LSM tree strikes a better balance between read and write performance and is adopted in many modern storage systems, including Google BigTable [10]/LevelDB [11], Apache HBase [12], Apache Cassandra [13], Facebook RocksDB [14], etc.

At a high level, an LSM tree lays out its storage into several "levels" and supports, in addition to reads/writes, a compaction operation that reorganizes the leveled storage for future read/write efficiency. We propose a scheme to log the LSM tree in Blockchain: 1) individual levels are digested using Merkle trees with the 128-bit root hashes stored in Blockchain. 2) The compaction that needs to be carried out in a trustworthy way is executed in smart-contracts, which allow for computations on modern Blockchain, such as Ethereum [2]. Concretely, we propose compaction mechanisms that realize several primitives inside the smart contract. We propose a duplicated compaction paradigm amenable for implementation on the asynchronous Smart-Contract execution model in Ethereum. Based on the primitives and paradigm, we realize both sized and leveled compaction mechanisms in the smart contract.

We have implemented the design on Ethereum leveraging its Smart-contract language, Solidity [15], and programming support in the Truffle framework [16]. In particular, invoking a Solidity smart-contract is asynchronous and our system addresses this property by asynchronously compacting the LSM storage. Based on the implementation, we evaluate the cost of our proposed scheme with the comparison to alternative designs. The results show the effectiveness of cost-reducing approaches used in our work.

The contributions of this work are the following:

1. We propose TPAD, a novel architecture to secure outsourced data storage over the Blockchain. The TPAD architecture considers an LSM-tree-based storage protocol and maps security-essential state and operations to the Blockchain. The architecture includes a minimal state in Blockchain storage and offline compaction operations in the asynchronous smart-contract.

2. We implement a prototype on Ethereum/Solidity that realizes the proposed design. Through evaluation, we demonstrate the effective cost saving of the TPAD design with the comparison to state-of-the-art approaches.

The rest of the paper is organized as following: § II formulates the research problem. The proposed technique, LSM-tree based storage over the Blockchain, is presented in § III. The system implementation is described in § IV. Evaluation is presented next in § V and § VII surveys the related work. § VIII concludes the paper.

II. Problem Formulation

A. Target Applications

This work targets the application of secure data outsourcing in a third-party host (e.g., Amazon S3). A particular scenario of interest is to outsource the data generated by the Internet of things (IoT) devices to the cloud storage, which serves read requests from data consuming applications. The IoT data is usually personal and could be security sensitive; for instance, in the smart home, the IoT devices such as smart TV controller capture residents' daily activities which could reveal personal secrets such as TV view habits. The IoT data, on the other hand, can be used to improve the life quality and enable novel applications. For instance, analyzing patient's activities at home can improve out-patient care and predict possible disease. In practice, various IoT data is widely collected and outsourced [17]. A noteworthy characteristic of our target application is that data is generated continuously and intensively. The workload is more write intensive than the static workload (e.g., in classic database systems).

B. System Model

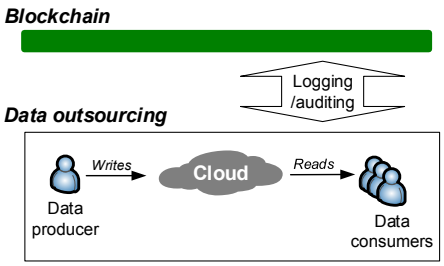


Fig. 1: Logging data outsourcing in Blockchain

The data-outsourcing system consists of data producers, a cloud host, and several data consumers. A data producer submits data write requests to the cloud and a data consumer submits data read requests to the cloud. The cloud exposes a standard key-value store interface for reads and writes. Formally, given key k, value v, timestamp ts, a data write and data read are described below:

$$
\begin{aligned}
ts &:= \mathsf{Put}(k, v) \\
\langle k, v, ts \rangle &:= \mathsf{Get}(k, ts_q)
\end{aligned} \tag{1}
$$

In our system, we assume the data producers and consumers are trusted. The third-party cloud is untrusted and it can launch various attacks to forge an answer to the consumer which will be elaborated on in § II-C.

Our data-outsourcing system has a companion of Blockchain, as illustrated in Figure 1. The Blockchain logs certain events in the workflow of data outsource for the purpose of securing it.

C. Security Goals

In the presence of the untrusted host, there are threats that could compromise data security. An adversary, be it the cloud host or man-in-the-middle adversary in networks, could forge a fake answer to a data consumer and violate the data integrity, membership authenticity, etc. The data integrity can be protected by simply attaching a message-authentication code (MAC) to each key-value record. This work considers the more advanced attacks — membership attacks that manifest in many forms: It could be the untrusted host deliberately skips query results and presents an incomplete answer (violating query completeness). It could be the host presents a stale version of the answer (violating query freshness). It could have the host to return different answers to different consumers regarding the same query (violating the fork consistency). On the write path, a man-in-the-middle adversary could replay a write request to result in incorrectly duplicated data versions. The formal definition of membership data authenticity is described in the existing protocols of authenticated data structures (ADS) [18], [19], [20], [21].

While this work mainly focuses authenticity, we consider a weak security goal w.r.t. the data confidentiality that deterministic encryption suffices. The extension for data confidentiality will be discussed in § VI.

D. Existing Techniques and Applicability

Existing works on **ADS construction**, while ensuring the security of membership authentication, are mostly designed based on read-optimized database structures such as B trees and R trees [8], [7] that perform data updates in place. These update-in-place structures translate an update operation from applications to a read-modify-write sequence on the underlying storage medium, and they are unfriendly to the write performance. The only ADS work we are aware of on address write efficiency is [22], which is however constructed using expensive lattice-based cryptography.

Without security, there are various write-optimized **log-structured data structures** that do not perform in-place updates but conduct append-only writes instead. A primary form of these data structures is to organize the primary data storage into a time-ordered log of records where an update is an append to the log end and a read may have to scan the entire log. The pure-log design is widely used in the log-structured journaling file systems [6].

A log-structured merge tree [9] represents a middle ground between the read-optimized update-in-place structure and the write-optimized log. An LSM tree serves a write in an append-only fashion and also supports random-access read without scanning the entire dataset. The LSM design has been adopted in many real-world cloud storage systems, including Google BigTable [10]/LevelDB [11], Apache HBase [12], Apache Cassandra [13], Facebook RocksDB [14], etc. The read-write characteristic of an LSM tree renders it well suited for the applications of IoT data outsourcing.

E. Motivation

Our target applications such as IoT data outsourcing feature a high-throughput stream of data updates and random-access read queries. As aforementioned, a Log-Structured Merge Tree is a good fit for this workload, assuming some offline hours for data compaction.

To map the LSM-tree workflow in an outsourcing scenario, it is essential to find a trusted third-party to conduct the data-compaction work. Relying on one of data owners to do the compaction is unfeasible due to availability, data owner's limited power (e.g., a low-end IoT device), etc.

We propose to leverage the Blockchain for the secure compaction in LSM storage. The decentralized design and large-scale deployment of existing Blockchain render it a trustworthy platform. The new smart-contract interface of the latest Blockchain makes it friendly to run general-purpose trustworthy computation on the platform.

Despite the advantages, designing a system for Blockchain-based LSM-storage outsourcing is non-trivial. Notably, Blockchain's innate limitation (in low storage capacity, high cost, low write throughput) presents technical challenges

when being adapted to the high-throughput data-outsourcing workflow. We address these challenges by limiting Blockchain's involvement in the online path of data outsourcing, such that the state on Blockchain can be "updated" infrequently.

F. Preliminary: LSM trees

The mechanism of an LSM tree is the following: It represents a dataset by multiple sorted runs (or files) and organized in several so-called "levels". The first level stores the most recent data writes and is "mutable". Other levels are immutable and are updated only in an offline manner. Concretely, a data write synchronously updates the first level. The first level may periodically persist data to an external place, called write-ahead log (WAL). When the first level becomes full, it is flushed to the next level. A read iterates over levels, and for each level, it is served by an indexed lookup. In the worst case, a read has to scan all levels and in practice, the total number of levels is bounded. In addition, if the application exhibits some data locality (i.e., reads tend to access recently updated data), a read can stop in the first couple of levels. An LSM tree supports a compaction[1] operation that merges multiple sorted runs into one and helps reorganizes the storage layout from a write-optimized one to a read-optimized one. The compaction is a batched job that usually runs asynchronously and during offline hours. There are two flavors in compaction, namely, flush and merge. A flush operation takes as input multiple sorted runs at level i and produces a sorted run as output at level $i+1$. A merge operation takes as input one selected file at level i and multiple files at level $i+1$ that overlap the selected file in key ranges. It produces sorted runs that replace these input files at level $i+1$.

An LSM mechanism supports different policies to trigger the execution of a compaction. These policies include sized configuration and leveled configuration: 1) In a sized configuration, each tree level has the capacity of storing a fixed number of sorted files, say K. The file size at level i is K^i (the first level has i to be 0). A flush-based compaction is triggered when there are K files filled in a level, say i. The compaction merges all K files at level i into one file at level $i+1$. With the sized-compaction policy, files at the same level may have their content overlap in key ranges, and a read has to scan all files in a level. 2) In a leveled configuration, any tree level is a sorted run where different files do not overlay in their key ranges. Data at level 0 is flushed to level 1 and data at level i, $\forall i \geq 1$, is merged to level $i+1$ [11]. A compaction can be triggered by application-specific conditions. A read within a level can be served by an indexed lookup without scanning.

III. LSM Data Storage over Blockchain

A. Baseline and Design Choices

Baseline: Our general design goal is to leverage Blockchain for securing data outsourcing. A baseline approach is to replace the cloud host by Blockchain. In the baseline, the Blockchain stores the entire dataset and directly interacts with the trusted clients of data producers and consumers through three smart contracts. On the write path, a "writer" contract accepts the data-write requests from the producers (encoded in the form of transactions) and sends them to the Blockchain. On the read path, a "reader" contract reads the Blockchain content to find the LSM tree level that contains the result. The Blockchain runs an offline "compaction" contract that is triggered by the same conditions of original LSM stores and that merges multiple sorted runs to reorganize the layout.

Design Space: The above baseline design raises two issues as below:

First, the baseline approach uses the Blockchain as the primary data storage, which is cost inefficient. Concretely, storing a bit in Blockchain is much more expensive and costly than storing it off-chain (e.g., in the cloud). A promising solution is to partition the LSM workflow and to result in a minimal and security-essential partition in Blockchain. This way, the primary data storage which is cumbersome is mapped off-chain to the cloud host.

Second, the baseline approach enforces a strong consistency semantic over the Blockchain which is weakly consistent; this mismatch across layers may present issues and incur unnecessary cost. More specifically, the current system of Blockchain promises only eventual consistency (or timed consistency [23]) in the sense that it allows an arbitrary delay between the transaction-submission time and the final settlement time (i.e., when the transaction is confirmed in the blockchain). The eventual consistency limits the use of Blockchain for real-time data serving and renders the baseline approach that aggressively checks the Blockchain digests to be ineffective.

[1] In this work, the words of "compaction" and "merge" are interchangeably used.

B. Blockchain-Based TPAD protocol

TPAD overview: Our proposed TPAD protocol addresses the partitioning problem of an LSM tree for the minimal involvement with the Blockchain. The TPAD design separates the "data plane" (the primary data storage) and the "control plane" (e.g., digest management), and maps the former to the off-chain cloud and only loads the latter in Blockchain. Recall that an LSM tree supports three major operations (i.e., data write, read and compaction). For online data reads/writes, TPAD places only in Blockchain/smart-contract the access of the digests, while leaving data access and proof construction off-chain. To address the consistency limits, TPAD embeds the weak-consistency semantics in the application layer; for instance, it does not access the Blockchain if the results are too recent to be reflected in the Blockchain. The data-intensive computation of compaction is however materialized inside the Blockchain, which simulates a multi-client verifiable computation protocol [24]. This subsection presents the details of the TPAD protocol.

Recall that our overall system includes data producers, the cloud, the blockchain, and data consumers. The data producers generate data records and upload them to the third-party cloud-blockchain platform. Data consumers query the cloud by data keys to retrieve relevant records. For the ease of presentation, we use a concrete setting w.l.o.g. that involves two data producers, say Alice (A) and Bob (B), and one data consumer, say Charlie (C).

Initially, each data producer has a pubic-private key pair and uses the public key as her pseudonymous identity. In other words, the system is open membership that anyone can join, which is consistent with the design of open Blockchain. We assume the identities of data producer and Blockchain are established in a trusted manner, which in practice could be enforced by external mechanisms for user authentication and attestation. Conceptually, there are two virtual chains registered in the Blockchain to materialize the two states of an LSM tree, that is, the WAL and digests of data levels. These two virtual chains can be materialized in the same physical Blockchain.

On the write path, Alice, the producer, generates a record (R_A) and submits it to the Blockchain through the logger contract that logs the record as a transaction in the WAL Blockchain. The logger contract is called asynchronously in that it returns immediately and does not wait for the final inclusion of the transaction in WAL Blockchain. Simultaneously, Alice also sends the record to the untrusted cloud, which stores it in Level 0 of its local LSM system. Bob sends another record R_B to the Blockchain and cloud, which is processed in a similar fashion. The logger contract is responsible for serializing multiple records received and sending transactions in order. The total order between R_A and R_B is not resolved until the transactions are finally settled in the WAL Blockchain, which could occur as late as up to 40 min (e.g., in BitCoin) after the submission time. We maintain the consistency semantics that there is no time ordering among records in Level 0 on the cloud. Upon flush, it only flushes the records whose transactions are fully settled in the Blockchain.

On the read path, Charlie submits a query to the cloud, which returns the result as well as query proof. In addition, Charlie obtains the relevant digests from the Blockchain. Specifically, the proof consists of the Merkle authentication paths of all relevant levels, that is, the level that has the answer (i.e., membership level) and all the levels (i.e., non-membership levels) that do not have the answer but are more recent than the membership level. The digests, namely Merkle root hashes, are obtained from Digest Blockchain. As aforementioned, the system does not provide membership authentication for data in Level 0.

On the compaction path, TPAD supports two relevant contracts for data flush and merge. For the flush, the flush contract is triggered every time there is a new block found in the WAL Blockchain. It flushes all the files/records at level i to a single sorted file at Level $i+1$. Inside the flush, the contract sorts the records at level i (which are originally organized in the time order), builds a digest of the sorted run, and sends it to the Digest Blockchain. At the same time, the off-chain cloud runs the flush computation that builds the sorted run locally.

For the merge, a separate contract merges multiple sorted runs into one run and places it at a certain level of the LSM tree. When a compaction contract runs, it validates all the input runs fed from the cloud using the digests stored in the Blockchain. It then performs the merge computation, builds a Merkle root hash on the merged run, and sends a transaction encoding the hash to update the Digest Blockchain. In the last step, the Digest Blockchain stores the digests of different LSM levels and the contract replaces the digests by those of the merged run. At the same time, the off-chain cloud runs the merge computation that builds the sorted run locally.

The two compaction contracts update the Blockchain state and have a companion computation going on the off-chain side. Given the delay to finally settle a transaction, we defer the time the updated state in Blockchain becomes available. For instance, even though the merge contract finishes the execution and sends the transaction, the off-chain data store will wait until the transaction is settled to activate the use of merged runs. The above two compaction contracts involve data-intensive computation and are executed at off-line hours. The specific triggering conditions are described next.

The algorithms in TPAD are illustrated in Listing 2.

Compaction-triggering policies: In TPAD, the policy that determines when and how to run a compaction is executed by the cloud host. As aforementioned, the off-chain cloud can opt for the sized LSM tree policy where the number of files per level is fixed and an overflowing file triggers the execution of flush operation. The off-chain cloud can also take the leveled LSM tree policy where application-specific condition triggers the execution of merge operations. In practice, the sized policy lends itself to serving time-series workloads where newer data does not replace older data.

In our implementation, an LSM level in the Smart Contract program is represented by an array in memory. The output is the digest of merged data which is stored persistently on Blockchain. Note that we do not store or send the merge data in Smart Contract to save the Gas cost.

1) Security Analysis: We consider a data-freshness attack where an adversary, e.g., the untrusted host, presents a valid but stale key-value pair as the result. That is, given a query $\mathsf{Get}(k, ts_q)$, it returns $\langle k', v', ts' \rangle$ that belongs to the data store, while there exists another more fresh key-value record $\langle k, v, ts \rangle$ such that $ts' < ts < ts_q$.

The LPAD scheme can authenticate the following two properties that establish the data freshness: 1) Result membership: Given a result record from a specific level (called result level), the LPAD scheme can prove the membership of the record in the level using the corresponding Merkle tree. That is, given query result $\langle k', v, ts \rangle := \mathsf{Get}(k, ts_q)$, LPAD can authenticate the membership of $\langle k', v, ts \rangle$ in the level it resides in (using the per-level Merkle tree) and hence the membership in the data store. 2) Non-membership of any fresher result. That is, the LPAD scheme can prove the non-membership of any record of the same queried key in levels fresher than the result level. Note that for a given key, levels are ordered by time.

In a query-completeness attack, valid result records are deliberately omitted. The completeness security is similarly provided by the LPAD scheme with the freshness security: In LPAD, the result completeness (i.e., no valid result is missed) in each query level can be deduced from that the leaf nodes in each per-level Merkle tree is sorted by data keys.

In a forking attack, different views are presented to different querying clients (presenting "X" to Alice and "Y" to Bob). The forking-attack security (or fork consistency) can be guaranteed by LPAD by that the Blockchain can provide a single source of truth for the dataset state, and any violation (by forking) can be detected by checking the result against the Blockchain state.

IV. Implementation on Ethereum

We have implemented the TPAD protocol over the Ethereum Blockchain which keeps two states: WAL and digests. The other players in the protocol, including the data producers, consumers, and the cloud, are implemented in JavaScript.

A data producer writing a record to the cloud triggers the execution of logger contract on Ethereum that computes the hash digest and sends a transaction wrapping the digest.

A data consumer submits a query by key to the cloud which returns the answer and proof. The data consumer inquires about the digests stored in the Blockchain by triggering the execution of a reader contract on Blockchain. The answer proof consists of authentication paths of Merkle trees from the cloud and is used to compare against the digests for answer verification. Note that we implement the reading of digests in a smart contract for the ease of engineering.

A compaction operation is implemented on both the cloud and Blockchain. Consider the compaction of two files (or sorted runs). First, the compaction smart-contract on the Blockchain takes as input the data stored in JSON on the cloud side and the digest hashes stored in the Blockchain. As mentioned, the compaction code validates the inputs based on the digests, conducts the merge computation by heap sort, computes the new digest of the merged run, and sends the transaction encoding the digest to the Blockchain. Second, the JavaScript program on the cloud side also runs the merge computation locally on the input files. It then replaces the input files in the local JSON store by the merged file. We choose this implementation (merge computation done on both cloud and smart contract), because the JavaScript runs the smart contract asynchronously (i.e., the call returns in JavaScript without waiting for the smart contract finishes the execution) and it saves bandwidth.

A compaction operation is implemented as a distributed process running on the both sides of cloud and Blockchain. When the cloud (or a cloud administrator) decides to merge the LSM storage, it first uploads the data to be merged to the Blockchain using a batch of transactions. Then, the cloud starts to run a local merge operation. Concurrently, the transactions sent by the cloud triggers the execution of a smart-contract that does the merge computation on the Blockchain based on the data sent earlier. The cloud and Blockchain is synchronized when the merge computations

on both sides end. Concretely, the cloud, once it finishes the local merge computation, will wait until being notified by the completion event of the remote merge on the Blockchain. On implementation, the cloud merge program is written in Javascript and the synchronization is realized using Promise [25], which is a multithreading support in Javascript. After the synchronization, the cloud proceed to replace the data by the merged data.

On the blockchain, the verifiable-merge smart contract is implemented as below: The compaction code validates the input data based on the digest on Blockchain, carries out the merge computation based on heap sort, computes the new digest of the merged run, and persists it into the Blockchain by sending a transaction.

The logger contract is triggered when a data producer uploads a record and its digest. The flush contract is triggered by a block in the Blockchain is found. The compaction contract is triggered by LSM compaction policies elaborated in the next section.

Implementation notes: The current version of Solidity (i.e., 0.4.17) does not support multi-dimensional nested array in a public function. We have to implement the array of digests as a one-dimensional array and interpret it as a two-dimensional array (by levels and files) manually in the program. To collect the Gas consumption in a view function (i.e., the function that does not change state), we call `estimateGas()` function. In our implementation, the JavaScript code runs smart contract functions through JSON ABI files generated by the truffle compiler [16]. The state overwrites in Ethereum/Solidity program has to be explicit and is realized by delete and "push" operations.

```
1  TPADContract{
2    uint[] WAL;
3    unit[] digests;
4    flush(){
5      while(block_found()!=true);
6        list l0=get_6th_block();
7      validate(l0);
8      l10=sort(l0);
9      digests.send_tx(digest(l10));
10   }
11   compact(list l1, list l2){
12     while(l1,l2=compact_policy());
13     validate(l1,l2);
14     l12=merge(l1,l2);
15     digests.send_tx(digest(l12));
16   }
17 }
18 class Client {
19   write(record r){
20     cloud.write(r);
21     WAL.send_tx(r);
22   }
23   read(key k){
24     result a, proof p=cloud.read(r);
25     d=digests.read_tx(a);
26     if(verify(a,p,d)) return a;
27   }
28 }
```

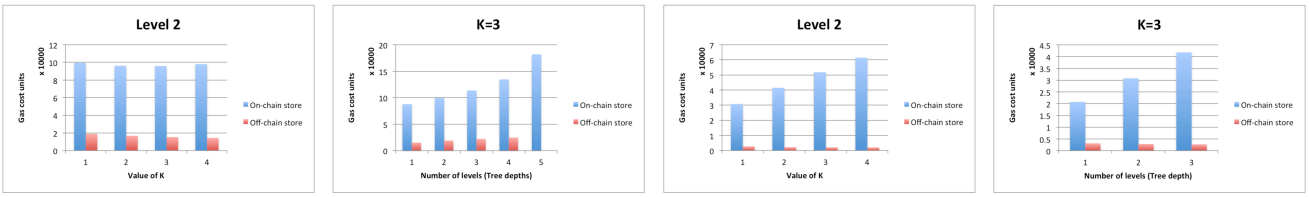
Fig. 2: Implementing TPAD

V. Evaluation

This section presents the evaluation of TPAD. The goal is to understand the cost saving of TPAD comparing alternative designs including on-chain storage (§ V-A) and other data structures (§ V-B). We first present our evaluation platform.

Setup: Our smart-contracts written in Solidity are compiled in the Truffle programming suit. They run on a personal Blockchain network set up by Ganache [26]. This local Blockchain network is sufficient for our evaluation purpose which only evaluates the cost consumption. For comparison, we implement the baseline approach of storing data in Blockchain. Here, the blockchain keeps a state of the LSM tree stored in a multi-dimensional storage array. In the implementation, no in-memory index is maintained and finding a record in a file is materialized by binary search. We also implement the other two baselines, namely append-only log and update-in-place structures. For the latter, we implement a binary-search tree and build a Merkle tree based on it with the root node stored in Blockchain.

A. Cost Saving of Off-chain Storage



(a) Write cost at level 2

(b) Write cost with $K = 3$ (K is the number of files per level)

(c) Read cost at level 2

(d) Read cost with $K = 3$

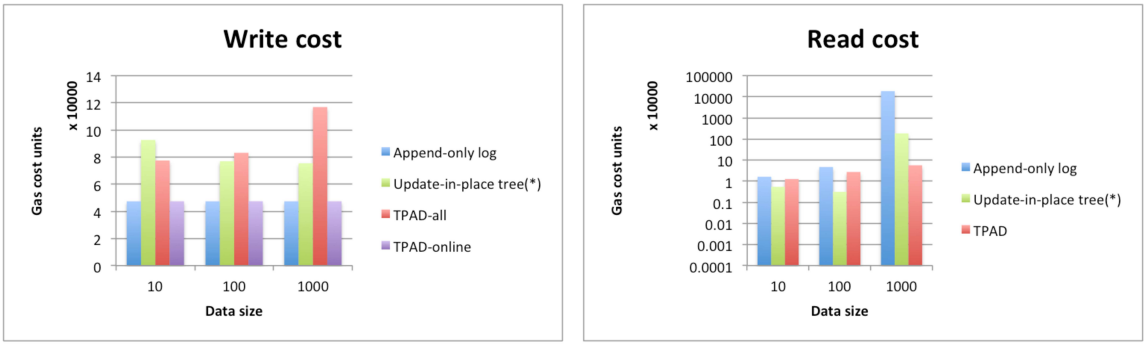
Fig. 3: On-chain storage cost versus off-chain cost

The TPAD is firstly a Blockchain logging scheme with the data stored off-chain. A relevant baseline is to treat the Blockchain as the primary storage, namely on-chain store. We implement the baseline by placing an entire LSM tree, including leaf-level data nodes, inside the Blockchain.

On our platform, we conduct experiments by driving $20,000$ records into the data store. We varied the "shape" of the LSM tree in terms of the size of a level (number of files allowed in a level, K) and the number of levels. We measure the cost in terms of Gas consumption of the two approaches respectively with on-chain and off-chain storage.

The results are presented in Figure 3. Figure 3a is the write cost when the LSM tree has two levels. With different values of K (recall K is the number of files in a level), the cost is relatively stable. Comparing the on-chain storage, the off-chain storage saves a significant amount of cost, which is about $5X$ saving. When fixing K at 3, varying the number of levels from 1 to 5, the cost of on-chain store increase which is consistent with the fact that write amplification increases along with the number of compaction jobs. Comparing on-chain and off-chain storage, the cost saving also increases along with the number of levels. In both Figure 3c and Figure 3d, the read cost increases along with the value of K. The off-chain storage saves the Gas cost up to $60X$ and $20X$ respectively for the settings of two levels and K equal to 3.

B. Efficiency of LSM-based Storage on Blockchain



(a) Write cost

(b) Read cost

Fig. 4: LSM tree-based TPAD compared against other structures in cost

The TPAD is an authenticated key-value store that supports random-access reads/writes. In this regard, relevant baselines that implement the key-value store abstraction include an append-only log where records are ordered by time and an update-in-place structure, namely a single Merkle tree where leaf nodes are ordered by keys. We implement the first baseline by simply sending the hash digest of every data write to the Blockchain. The second baseline is implemented by maintaining the root hash of the key-ordered Merkle tree in Blockchain and by translating every data read/write to a leaf-to-root path traversal on the Merkle tree. In more details, a data read to the cloud store would present as a proof the authentication path of the leaf node to the root hash of the Merkle tree

and a data write consists of a data read followed by a local modification and a remote update to the authentication path.

We conduct small-scale experiments by loading a thousand records into the storage system; the keys and values in the records are randomly distributed. We measure the average costs of read and write. The cost consists of the Gas cost for running smart contract that retrieves the digests stored in the Blockchain and the costs of preparing and verifying query proof (e.g., the authentication paths in Merkle trees). We use a heuristic to combine the two costs by multiplying the Gas cost by 100 times before adding it and the proof-related cost. The proof-related cost is measured by the number of cycles spent locally for proof verification.

The results are presented in Figure 4. The results show that the TPAD can result in cost efficiency on both reads and writes. Concretely, for the write results in Figure 4a, the online part of TPAD has a similar cost with the other two baselines, as each write results in a single transaction in all three approaches. The overall TPAD approach that includes both online and offline operations (i.e. compaction) would incur write amplification as shown in Figure 4a. For the data read results in Figure 4b, the cost saving of TPAD is significant, provided that the y-axis is plotted in log scale. The TPAD incurs even lower cost than update-in-place trees partly because of the locality in our query workloads where recently updated data is more likely being queried.

VI. Discussion: Data Confidentiality & Key Management

Data producers concerned about data confidentiality can upload the records in an encrypted form. Specifically, a data producer sends the ciphertext of the record, instead of plaintext, to the third-party host. The decryption key is shared through an offline key-distribution channel between the data producer and the data consumers who are permissioned to access the record. Those consumers can obtain the ciphertext of the record from the host and use the key to decrypt. To enable the query over ciphertext, we consider the use of deterministic encryption which supports exact-match query in the encrypted form, that is, the consumer could submit the encrypted query key to the host who will conduct exact-match query between the query ciphertext and data ciphertext. The integration with more secure encryption primitives is complementary to this scope of this work.

The data-encryption layer is laid over the membership-/data- authentication layer of TPAD (as described above). This is similar to the classic encryption-then-authentication scheme [27]. With deterministic encryption, the merge operation of TPAD occurs in the domain of ciphertext.

VII. Related Work

A. Blockchain Applications

A common paradigm of supporting applications over Blockchain is that the application-level workflow is partitioned and mapped to the on-/off-chain parts. Decentralizing privacy [28] supports access-control oriented data-sharing applications over Blockchain. It publishes the access control list onto the Blockchain and enforces the access control by smart contract. A similar approach is used in MedRec [29] to enforce access control for medical data sharing. MedRec runs a proprietary Blockchain network where miners are computers in an academic environment and are rewarded by an anonymized medical dataset.

Namecoin [4] and Blockstack [30] support general-purpose key-value storage in the decentralized fashion. They allow open-membership and accept any users to upload their data signed with their secret keys. They support the storage of name-value binding, with a canonical application to be DNS servers. Namecoin is a special-purpose Blockchain system and Blockstack is realized as a middleware on top of any Blockchain substrates. The VirtualChain in Blockstack supports a (single) state-machine abstraction. Re-purposing original Blockchain for storage, its system design tackles the challenges of limited storage capacity, long write latency, and low transactional throughput.

Catena [31] is probably the closest related work to TPAD. Catena is a non-equivocation scheme over the Blockchain that repurposes its no-double-spending security for non-equivocation in logging and auditing. In essence, it aligns the application-specific log (for auditing) with the underlying linear Blockchain and reuses the non-fork property of Blockchain for the non-fork application log. Briefly, Catena's mechanism is to build a virtual chain on BitCoin blockchain by (ab)using `OP_RETURN` transaction interface. Logging sends a BitCoin transaction and auditing performs an ordered sequence of statement-verification calls in the log history. The statement verification does not scan full history but simply runs the Bitcoin-validation logic (e.g., Simplified Payment Validation), which ensures no BitCoin double-spending. Importantly, it enforces the rule that a Catena transaction spends the output of its immediate predecessor for efficient validation. The genesis transaction is served as the ground truth of validation and it assumes a broadcast channel to establish the consistent view of the Genesis transaction.

Our TPAD is different from Catena in the following senses: 1) Catena is built on Bitcoin or the first-generation blockchain, and TPAD leverages the smart-contract capabilities widely existing in the latest Blockchain systems, such as Ethereum [2]. 2) More importantly, Catena only supports auditing which is essentially sequential reads. TPAD supports verifiable random-reads. 3) While Catena claims to be low cost, the increasing rate of BitCoin ($700 per BitCoin at the time of Catena paper writing versus $17000 per BitCoin at early 2018) makes the Catena more expensive. TPAD address the cost minimization of these repurposed Blockchains.

B. Outsourced Storage and ADS

Outsourcing data storage to a third-party host such as public cloud is a popular application paradigm. In the presence of an untrusted host, it is important to ensure the data security, especially membership authenticity. An authenticated data structure (ADS) is a protocol that formally the security property. Depending on the operations supported (queries and updates), an ADS protocol can be constructed by different cryptographic primitives such as secure hash and Merkle trees [32], SNARK [33], bilinear pairings [20], [34], etc.

VIII. Conclusion

This work proposes the TPAD system for securely outsourcing data storage on third-party hosts by leveraging the Blockchain. Instead of using Blockchain as a time-ordered log, TPAD lays the log-structured merge tree (LSM tree) over the Blockchain for efficient and lightweight logging. Realizing the design, a small state is persisted in the Blockchain and computation-oriented compaction operations are carried out by smart contracts. With the implementation in Ethereum/Solidity, we evaluate the proposed logging scheme and demonstrate its performance efficiency and effectiveness in cost saving.

References

[1] "Bitcoin: https://bitcoin.org/en/."
[2] "Ethereum project: https://www.ethereum.org/."
[3] "Litecoin: https://litecoin.org/."
[4] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in 14th Annual Workshop on the Economics of Information Security, WEIS 2015, Delft, The Netherlands, 22-23 June, 2015, 2015. [Online]. Available: http://www.econinfosec.org/archive/weis2015/papers/WEIS_2015_kalodner.pdf
[5] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction. Princeton University Press, 2016. [Online]. Available: http://press.princeton.edu/titles/10908.html
[6] M. Rosenblum, The Design and Implementation of a Log-Structured File-System. Kluwer, 1995.
[7] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 2nd Edition. Benjamin/Cummings, 1994.
[8] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 5th Edition. McGraw-Hill Book Company, 2005.
[9] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil, "The log-structured merge-tree (lsm-tree)," Acta Inf., vol. 33, no. 4, pp. 351–385, 1996.
[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data (awarded best paper!)," in OSDI, 2006, pp. 205–218.
[11] "Google LevelDB, http://code.google.com/p/leveldb/."
[12] "Apache HBase, http://hbase.apache.org/."
[13] "Apache Cassandra, http://cassandra.apache.org/."
[14] "Facebook RocksDB, http://rocksdb.org/."
[15] "Solidity: https://solidity.readthedocs.io/en/develop/."
[16] "Truffle: http://truffleframework.com/."
[17] H. Chung, M. Iorga, J. M. Voas, and S. Lee, ""alexa, can I trust you?"," IEEE Computer, vol. 50, no. 9, pp. 100–104, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.3571053
[18] R. Tamassia, "Authenticated data structures," in Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings, 2003, pp. 2–5. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-39658-1_2
[19] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables based on cryptographic accumulators," Algorithmica, vol. 74, no. 2, pp. 664–712, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00453-014-9968-3
[20] Y. Zhang, J. Katz, and C. Papamanthou, "Integridb: Verifiable SQL for outsourced databases," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015, 2015, pp. 1480–1491. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813711
[21] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in SIGMOD Conference, 2006, pp. 121–132.
[22] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, 2013, pp. 353–370. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38348-9_22
[23] D. Terry, "Replicated data consistency explained through baseball," Commun. ACM, vol. 56, no. 12, pp. 82–89, 2013. [Online]. Available: http://doi.acm.org/10.1145/2500500

[24] S. D. Gordon, J. Katz, F. Liu, E. Shi, and H. Zhou, "Multi-client verifiable computation with stronger security guarantees," in Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II, ser. Lecture Notes in Computer Science, Y. Dodis and J. B. Nielsen, Eds., vol. 9015. Springer, 2015, pp. 144–168. [Online]. Available: https://doi.org/10.1007/978-3-662-46497-7_6

[25] "Promise: https://developer.mozilla.org/en-us/docs/web/javascript/reference/global_objects/promise." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[26] "Ganache: http://truffleframework.com/ganache/."

[27] J. Katz and Y. Lindell, Introduction to Modern Cryptography. Chapman and Hall/CRC Press, 2007.

[28] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in 2015 IEEE Symposium on Security and Privacy Workshops, SPW 2015, San Jose, CA, USA, May 21-22, 2015. IEEE Computer Society, 2015, pp. 180–184. [Online]. Available: https://doi.org/10.1109/SPW.2015.27

[29] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in 2nd International Conference on Open and Big Data, OBD 2016, Vienna, Austria, August 22-24, 2016, I. Awan and M. Younas, Eds. IEEE Computer Society, 2016, pp. 25–30. [Online]. Available: https://doi.org/10.1109/OBD.2016.11

[30] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in 2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016., A. Gulati and H. Weatherspoon, Eds. USENIX Association, 2016, pp. 181–194. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/ali

[31] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. IEEE Computer Society, 2017, pp. 393–409. [Online]. Available: https://doi.org/10.1109/SP.2017.19

[32] R. C. Merkle, "Protocols for public key cryptosystems," in IEEE Symposium on Security and Privacy, 1980, pp. 122–134.

[33] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for C: verifying program executions succinctly and in zero knowledge," in Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II, 2013, pp. 90–108. [Online]. Available: https://doi.org/10.1007/978-3-642-40084-1_6

[34] Y. Zhang, J. Katz, and C. Papamanthou, "An expressive (zero-knowledge) set accumulator," in 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. IEEE, 2017, pp. 158–173. [Online]. Available: https://doi.org/10.1109/EuroSP.2017.35

Comparative Analysis of the Legal Concept of Title Rights in Real Estate and the Technology of Tokens: How Can Titles Become Tokens?

Oleksii Konashevych

Erasmus Mundus Joint International Doctoral Fellow in
Law, Science and Technology, a.konashevich@gmail.com,
Supervisors: Prof Marta Poblet Balcell and Prof Pompeu Casanovas

Abstract.
This paper discusses how to use blockchain tokens to represent real estate titles. Tokens on the blockchain as a technological concept is the closest solution to the legal concept of titles, because it provides for evidence of ownership and can be transferred from one address to another, while giving exclusive access to such an address to the owner. This paper contains the analysis of the concept of tokens in the context of its applicability to title rights on real estate. There is also a discussion of the outcomes of conducted interviews with professionals in the field of Computer Science, technologies, blockchain and smart contracts. Some critical mismatches were found: tokens are not able to satisfy current demand to manage title rights online. To develop a mature and sustainable electronic system, there are certain issues that need to be addressed: inheritance procedures, litigation, guardianship, delegation of rights and rights of third parties (liens and encumbrances) as well as the legal concept of bundle of rights (possession, disposition, enjoyment, etc.), which requires a strong mathematical model. During the abovementioned interviews, some weaknesses were found in the existing ideas of the use of the blockchain for real estate, mostly related to the undesirable centralization and issues with security. As the result of this research, it is obvious what needs to be developed is the concept of a high-level design of the technology, capable of managing title rights on the blockchain, which includes a three-level mechanism of 1) e-voting, which provides for a democratic implementation of governing algorithms; 2) Smart Laws, as the concept of high level "smart" algorithms that implement (by e-voting) existing laws related to property rights in a form of the program/protocol; and 3) smart contract templates which are based on the smart laws, that allow people to manage their title rights online.

Keywords: Blockchain, Smart Contracts, Titles, Real Estate, Tokens, E-Governance, E-Voting, E-Democracy.

1 Introduction

A [crypto] token is a record of a number which is kept by a specific address on the blockchain and can be divided (usually up to 8 decimals) and transferred to another address within the ledger of the blockchain system [1], [2]. For the purposes of this paper, we do not distinguish between tokens and cryptocurrency. However, tokens have more features and may be considered as a technological evolution of the cryptocurrency, presented by someone who called himself Satoshi Nakamoto [1].

A few principal features make tokens ideal for the management of property rights:

1. The blockchain protocol is designed to make transactions – the transfer of tokens from one address to another, while not allowing double spending [1], [3].

2. Such address provides for exclusive access, because only a person who has a cryptographic private key may manage the token.

3. The blockchain ledger is a complete, transparent history of records, which allows a person to track each token from the moment of creation, including fractional transactions (decimals and less than 1) and transactions between any number of addresses.

The three next features make tokens principally different and more developed compared to a traditional, centralized way of making ledgers that is typical for banks and public registries:

1. The technology offers a decentralized way of keeping records; no one keeps all of the power in his hand, and that prevents usurpation of power and corruption.

2. The immutability and the non-returnability of transactions, which means that it is practically unfeasible to delete or alter a record or in any other way to corrupt it.

3. The next generation of blockchains (after Bitcoin) offers algorithms to introduce a high level of automation and security for the management of tokens and at the same time, excludes the necessity of a human to operate it manually. (See all critical features in a diagram on Fig. 1.)

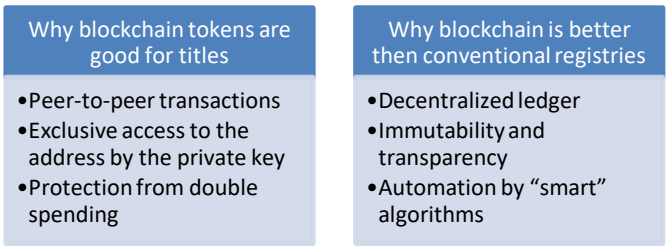


Fig. 1 Critical features of the blockchain for the managing of titles

Ethereum [4] and NXT [5] are examples of blockchain-based platforms that develop such automated systems. However, they stand on different ideological grounds. Ethereum is a Turing-complete programming language platform [6] for the developing of so-called "smart contracts" [7], and NXT suggests ready-to-use user services

implemented into the core of the blockchain protocol with no need to develop applications.

The mentioned features of the blockchain technology with smart contracts can be considered as an alternative technology to existing centralized land (real estate/cadastral) registries and a way to protect and manage title rights.

However, technological concepts appeared to be non-synchronized with legal concepts. In the following sections, it is shown why titles are not tokens, and what should be developed to introduce the world to a sustainable electronic system, able to improve the way how relations on real estate ownership is organized now.

2 Comparative Analysis of the Legal Concept of Title Rights and the Technology Tokens

The legal concept of titles and the technology of tokens have much in common, along with differences.

A "title" is an evidence of ownership. The title represents the property rights of an estate; this is an equivalent of estate (land, for example), but on paper, which is legally recognized. The crypto-token is a technology that has the same purposes – the token can represent values and prove ownership. The only difference is that titles of real estate have a long tradition and legacy of regulation, and there is no place for tokens in the existing laws. That is why transactions that are made with real estate tokens will not have any legal consequences.

Title deeds must be *acknowledged* in some countries before a notary, in some - before other authorized persons, and *recorded* in the public registry. Thus, the use of tokens for real estate requires legislative changes that legitimize new procedures of acknowledgement and recording on the blockchain.

The title can be divisible, that is, what in the language of law means co-ownership, joint ownership, community property (also known as marital property) and some other concepts that exist in different jurisdictions. There are two main aspects of property rights: the type of ownership, and a set of specific rules which co-owners must follow to respect the rights of other co-owners.

In the theory of law, there are two basic types of co-property: it may belong to persons on the right of common share, or on the right of common joint ownership. In common share, there are no fractions; the property belongs to all co-owners equally (spouses, condominium owners, etc.). In joint ownership, owners have shares (1/2, 1/3 etc.).

As to a set of rules, the law and the agreement may establish some specific rules which co-owners must follow. For, example, there is a typical rule that one co-owner cannot convey his share in the title without the consent of the other owners. The other owners have the right to buy the share for the same price as the owner wants to sell it.

The common property may become joint ownership. In the case, for example, that spouses divorce in some jurisdictions, they become 50/50 co-owners.

Different jurisdictions may have some specifics in co-ownership law, as well as individual may have agreements between co-owners to establish specific rules. All

these rules in general can be represented in the theory of "property rights," which is further discussed.

That is why when we are talking about tokens, it is clear that at least two layers of technology solutions must be applied to tokens: the first is a set of algorithms that establish general rules (laws) specific for certain jurisdictions, and the second, individual rules based on contracts, that do not contradict general rules.

However, co-owners are not the only category of third parties that can influence the property rights of an owner. There are two other categories of third parties:

- third parties which are not owners but have interests in the property (the property rights of third parties) as per the law or agreement; and
- third parties that have no interests but have legal access to the property and may influence it (judge, notary, parents, custodian, town's clerk (registrar) etc.).

The concept of property rights includes a bundle of rights: the right to dispose, the right to possess and the right to use (enjoy)[1]. The owner is free to manage these rights and deeds that he concludes influence this bundle (See Fig. 2 "Components of ownership: bundle of rights".)

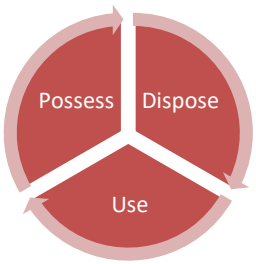


Fig. 2. Components of ownership: bundle of rights

For example, when an owner rents out his property, he transfers his right to possess and own the property to the third party – the tenant. At the same time, he as a landlord is restricted in these rights (to possess and to use) while the contract is valid. He is also restricted in his right to dispose of the property in the sense of allowing the use of the property by others. However, he keeps the right to convey the title (for example, to sell it). So, if he sells the property, the tenant keeps the rights to possess and to use the estate (unless otherwise provided by the contract), and a new title owner is granted with the same restrictions. There also can be other limits to dispose: in a mortgage, the owner is not able to convey the title without the agreement of the creditor.

As we see, the concept of property rights is complex, and the situation is more complicated by the existence of different jurisdictions and traditions of law. We see an essential need to present a mathematical model of property rights that matches the

[1] There are a couple of main theories about property rights, and they vary from 3 to 5 main rights: possession, disposition, enjoyment (use), control, exclusion etc. However, they do not have principal differences for this stage of research, and so it is not critical to make a choice now which theory fits best to design the technology. Our aim here is just to argue that these kinds of legal concepts create the necessity to find a solution to develop the technology.

concept of tokens driven by smart contracts, taking into account all specifics the comes from the blockchain technology, i.e. immutability of records and smart contracts, so all necessary high-level features must be developed by design not on the run. So, the mathematical model will become a metamodel for designing systems oriented on certain legal systems and jurisdictions.

There is also another category that has no interests in property, but acts in the interest of owners and other persons and may change property rights.

The judicial system allows interested parties to contest rights in court, so judges and then bailiffs are those who can change the title and property rights and enforce these things.

Another important group of third parties is a notary public. Notaries execute wills and apply inheritance laws.

Another case of the disposition of estate without the will of an owner are parents' rights, guardianship, and custodianship in respect of the rights of minors and disabled persons.

Parents, guardians, and custodians have the rights to act in the name of minors and disabled persons, including the right to dispose of estates and acquire property rights. Their rights may be unlimited or regulated by law and revised by a public body (custodian committees/boards, etc.).

The delegation of rights by contract is another case when the title is operated by the third party. In contrast to parents/custodians this delegation is not by law but by an agreement. The exact volume of rights is defined by law and by a contract and usually is confirmed by the power of attorney.

The last case when a title is under the influence of third parties, is when it is under the obligation to obtain permission from the public body to convey the property. There can be variety of reasons to do so. In this way, a government can:

- prevent illegal construction on the land;
- enforce owners to pay taxes before selling the property;
- oblige an appraisal of real estate (for some categories of owners, like state enterprises); or a
- local community may protect its right to not let unwanted people live in their territory.

Therefore, the approval and certain legal actions must be performed before a deed.

Another case which requires further development is the separation and merging of titles. That happens when adjoined plots of land are united into one title or a plot is divided. So, the mechanism to separate and merge the tokens by linking them to a new survey[2] is also required for a prospective electronic system.

None of this is implemented in the existing electronic solutions, and yet needs to be designed in the system that aims to provide a full range of legal instruments to manage property rights by smart contracts.

---

[2] Surveying or land surveying is the technique of determining the terrestrial or three-dimensional positions of points and the distances and angles between embodied on the plan of the plot.

3 The High-Level Design of The System

In preparation for this paper, 4 interviews were conducted and an analysis was made of existing projects related to the blockchain and real estate, and some typical cases of ICOs.

Those interviewed were: 1) Vassilis Vutsadakis, Ph.D. in Science and Technology, Researcher at Propy[3] (Blockchain Supermarket for Real Estate); 2) Matt McKibbin, Masters of Science in Industrial Hygiene, Co-founder and Business Development Director of "Ubitquity, LLC."[4] (the blockchain-secured platform for real estate recordkeeping), Founder and Chief Decentralization Officer of "DecentraNet" (blockchain consulting); 3) Mykhailo Tiutin, Masters in Information Security, Co-Founder/CTO of "Vareger" (Blockchain Developer, Ukraine)[5], smart contracts and blockchain developer, cryptographer and IT security expert; 4) Vadim Sukhomlinov, Masters in Computer Science, Software Engineer/Architect at Intel Corporation.

The research includes the analysis of projects that develop different solutions in blockchain and real estate domain.

- Velox.re[6] (USA, since 2016) is ongoing startup that aims to digitize the process of purchase in Cook County on Illinois state by creating a Bitcoin based platform that unites professionals (intermediaries) of real estate industry around the world [8], [9], [10].
- Ubitquity.io[7] (USA, since 2015) develops services on e-recording companies, title companies, municipalities, and custom clients to record of ownership [11].
- Bitland[8] (Ghana, since 2016) is a partner of Ubitquity.io and currently is developing solutions for Real Estate Land Registration services to citizens of Ghana as well as companies and farm unions [12], [13].
- Chromaway[9] (Sweden, since 2016) is piloting the first project to model a property purchase using the blockchain and smart contract technology [14].
- Flip[10] (USA, since 2016) is a peer-to-peer leasing marketplace in New York city [15];
- REX[11] (USA, since 2016) is a peer to peer MLS[12] built on Ethereum. REX aims to connect vendors, buyers and agents over an open network by using the blockchain cryptocurrencies [16].

---

[3] http://propy.com/
[4] https://www.ubitquity.io/web/index.html
[5] https://vareger.com/
[6] https://www.velox.re/
[7] https://www.ubitquity.io/web/index.html
[8] http://bitlandglobal.com/
[9] https://chromaway.com
[10] https://flip.lease/
[11] http://rexmls.com/
[12] MLS is a standard of listing real estate and services of brokers, http://www.mls.com/

- Bitfury[13] (Republic of Georgia, since 2016; Ukraine, since 2017) proposed a solution to protect records of the central cadastral registry by casting hashes of such records to the blockchain, mixing by fact two blockchain technologies: private DLT[14] EXONUM and Bitcoin [17], [18].
- Xinyuan Real Estate Co.[15] (China, since 2016), a company that declared its interest in the blockchain in the cooperation with IBM to develop a smart city in China [19].
- Propy, Inc.[16] (USA, since 2016), a company that develops the supermarket for real estate and platform for deeds based on Ethereum smart contracts.

Now let us summarize the aforesaid and the first section and discuss how to design the system in the best way. While doing interviews and researching, we distinguished some solutions which are not acceptable from our point of view. So, let us discuss first what we should not do and why.

The first is to use the blockchain as a database of records that reflects acts made offline. The blockchain in this case in not a primary source of evidence, but collects everything that is happening offline (on papers) or in the central database. Each new record is not necessary valid, but it helps to find the truth in a court while considering all possible paper evidences. We don't see much benefit in using the blockchain in this way because the central public registry does the same. The only thing the blockchain does is it protects against altering records, while a well-designed and protected central database can do the same. What is more important here, is that it does not require changes in the existing bureaucratic systems. But our aim is to find a better system that can reduce regulations and manual work.

Another sub-option of this approach is just to store hashes on the blockchain of records made in the central database, which is almost the same, but does give more protection to the database against corruption, and adds a new bureaucratic procedure [20].

In this concept, a private company or a public body keeps copies of private keys, and/or use multi signatures (escrow mechanism) [21]. In case the token is stolen, the company will announce it invalid and reissue a new token (we remember that we cannot alter the transaction, so if it is stolen we cannot do anything), so the company needs to manually track the list of tokens[17] and its validity. We see that this concept does not bring much value compared to the existing approaches because it is still centralized. There are too many examples of even large and well secured companies being hacked and losing personal data. A private company can lose not only the pri-

[13] http://bitfury.com/

[14] DLT is a Distributed Ledger Technology which means shared ledger technologies similar but not equal to the blockchain

[15] http://www.prnewswire.com/news-releases/xinyuan-real-estate-co-ltd-announces-blockchain-powered-real-estate-finance-technology-platform-300299818.html

[16] www.propy.com

[17] During the research we also found some ideas not to use tokens, but only to make deed records on the blockchain. However, the same as with tokens it requires a third party manually to track all legal facts which are occurred with the title and reflect its validity in case it has been recognized as invalid.

vate keys of users, but also their own keys; they can also be corrupted or even become bankrupt, which is especially an undesirable risk for people whose real estate is the only wealth they have.

In the case that we use the public body instead of a private entity, we will have more trust and more authority, but at the same time, we will create the same high-level regulations and bureaucracy.

Another arguable solution found was a creation of an electronic compliance system. For each transaction of a token, the owner uses a specific smart contract. As we remember, the smart contract is not a contract in the common sense, but just an electronic algorithm. For example, for a purchase: the program holds the transaction of the token until the buyer pays.

In real life, the contract is not a self-sufficient and closed legal act. The contract reflects the agreement of parties as to essential conditions, but laws at the same time provide for norms that are not necessarily included in the agreement, and are followed as if they were in the contract.

Sometimes it is almost impossible to include all of the provisions in the contract. So, the contract may only refer to the law, or even just presume that a law or a general practice will be applied to a missing part of the contract. A smart contract, which is a sort of a closed system, in this sense is flawed because cannot be influenced by external factors (like the law).

One solution is to use electronic compliance systems. Before a transaction, the compliance system will verify the token and the parties. In this case, we must ensure that such a system is good enough to protect the rights of parties according to the local jurisdiction and is not corrupted, which bring us similar issues to that of the previous example with the private company that manages keys and records.

Considering these items, we see that the best way to proceed is to develop the system so that the government will adopt it, is to implement by design existing specifics of jurisdictions according to the concept "code is law."[18] The code implements required provisions from the legislation, and in case something goes wrong, parties will use mechanisms of litigation and arbitration.

Algorithms adopted by the government will be a higher layer for smart contracts and will work as obligatory standards. Let us call these "smart laws." Smart laws will establish rules and mechanism of access of third parties to tokens and some basics principles of work (that reflect existing "paper" regulations).

Combing this system with the concept of oracles[19], which is proposed by V. Buterin [22], we will be able to keep track of authorized persons: the list of addresses of public notaries, judges, bailiffs, custodians, etc., that may perform transactions.

---

[18] The expression "code is law" was proposed by Lawrence Lessig in his Book "Code and Other Laws of Cyberspace" (1999)

[19] i.e. special servers, from which a smart contract receives reliable information from outside the smart contract

Any smart contract designed based on these smart laws will be able to provide the whole range of legal instruments, and if the situation with ownership and property rights is stuck, parties will be able to settle it in a court.

For example, the smart contract does not "know" when the owner dies; that is why we need an oracle that tracks records on a public demographic registry, and will trigger inheritance mechanisms of the smart contract. In this case a "smart will" would be executed.

If the person did not leave a will, general rules "smart laws" will be applied. The smart contract also does not "know" which notary will manage the distribution of the inheritance. That is why the oracle will provide the valid list of addresses of notaries, and only a transaction that comes from the address on the trusted list will be executed by the system.

Smart laws will provide necessary rules to run public oracles. Oracles require manual management: someone must add records in the demographic registry, update the notaries list, the custodians list, etc. But now this is performed by the government anyway; the only question is how well enough it is digitized and protected from corruption and fraud.

Oracles assume a certain degree of centralization or at least we cannot think of it as a pure distributed system (as the blockchain is) because it requires actions of third parties. The fact is that it is merely possible with reasonable efforts on this stage of development of science and technology to automate and digitalize everything. For, example, how to digitize the fact of human's death and make it a system event that triggers smart contract execution? Someone must certify plenty of facts that occur in real world that have legal meaning for property rights.

The centralization is not a threat it is only an environment where risks of corruption and excessive regulations arise from. Therefore, the question is how such oracles are well designed to protect from these risks.

To protect smart laws that run oracles from the corruption, they must not allow any backdoor access of someone specific to change them. The code, once deployed, must remain unchanged. And here the blockchain plays a significant role, because as we see with the example of a "smart contract," we can deploy completely transparent and verifiable applications protected from someone's manual control.

These closed, decentralized applications can work permanently secured from an alteration, and this is a benefit and a limit at the same time. The only way to change something here is to change the code of the blockchain protocol, which as we know requires a large consensus (usually 50+1% of nodes must support a "hard fork").

However, we still must have access to update the system. That is why at the upper level will be algorithms of electronic voting on the blockchain. Voting will be a public democratic mechanism of the control over smart laws systems and protect them from the corruption.

If someone's token is stolen, an authorized third party from the public oracles list will bring back access. But in case any of the public oracles are compromised, i.e. hacked, or keys are lost, or similar threats when oracles become technically uncontrolled or controlled and corrupted by unauthorized persons, the general public and/or

specially governed body (committee) by the voting mechanism will recall and reissue private keys to the operator of the oracle.

Finally, we have the mechanism of 4 layers of a democratic electronic governance (see Fig. 3):

- **Blockchain**. On the top we have a public blockchain which protocol remains unchanged by the consensus of node owners. It is important to have as many as possible of the active citizens that share their resources to the network. Good Samaritans will give a critical mass of consensus that will not allow changing of the protocol. The software and the consensus mechanism must be affordable to allow as many as possible "good Samaritans" to have their nodes. In this sense Proof-of-Work is not good, since the mining rush leads to high costs of entrance into the business. And of course, the blockchain must be public, so anyone can become a part of the network and receive crypto currency for its work.
- **E-Voting** is an irrevocable mechanism for voting on implementing smart laws, as nobody can change this mechanism, except to change the blockchain protocol (which requires consensus). Ballots are recorded on the blockchain, and the result of the voting automatically triggers mechanism of implementation of a smart law.
- **Smart laws** – the mechanism that controls public oracles and other basic mechanisms of the operating of tokens. Oracles keep lists of addresses that are authorized to change the status of smart contracts (judicial system, public notary, social system of custodian and guardianship).
- **Smart contracts** (templates of smart contracts) – owners will be able to use specific templates for their tokens (smart will, purchase, rent, mortgage). Each jurisdiction can use specific rules to introduce smart contracts. One of the possible scenarios is that in each state, local professionals (lawyer, notaries etc.) and IT developers will develop necessary templates of smart contracts in accordance with local jurisdiction. However, another way is when the government takes this process in its hands and introduces smart contract templates as model solutions (similar to "Model Company Charter"[20]).

[20] In many countries, governments adopt a "Model Company Charter" which people may use when they list a new company, so they do not need to write articles of incorporation (statute, charter) from scratch, but just to refer to this model paper which they submit as the official application to a registrar.

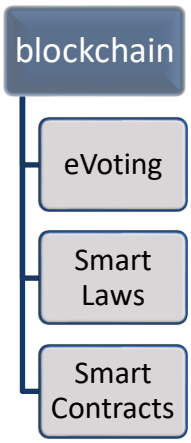


Fig. 3. The layers of the blockchain governing

4 Conclusion

Tokens on the blockchain as a technological concept is the closest solution to the issues of the legal concept of real estate titles. This paper distinguished the principal features that make the blockchain suitable for title rights and transactions: tokens can represent property rights, the technology protects from double spending while allowing for conveyance, addresses that store records of tokens are designed in a way to provide for exclusive access to such addresses to the owner and immutably of records which protect from corruption and fraud. The blockchain technology addresses the inherent issues of conventional governing since it works in a decentralized manner. With the second generation of blockchains that integrate smart contracts and similar algorithms, one can automate tokens/titles management, which reduces the participation of third parties (brokers, notaries, agents) or even governments to manually control the relations in the real estate domain. All these features make the technology applicable to significantly improve relations in real estate and governance.

This paper presented analysis of the concept of tokens in the context of its applicability to the legal concept of title rights on real estate, there is also a discussion of the outcomes of the conducted interviews with professionals in the field of Computer Science, technologies, blockchain and smart contracts. It was found some critical mismatches: tokens are not able to satisfy current demand to manage title rights online. To develop mature and sustainable electronic system there are certain issues to be addressed: inheritance procedures, litigation, guardianship, delegation of rights and rights of third parties (liens and encumbrances) as well as the legal concept of bundle of rights (possession, disposition, enjoyment, etc.), which requires strong mathemati-

cal model. During interviews it was found some weaknesses in existing ideas of the use of the blockchain for real estate: mostly related to the undesirable centralization and issues with security. It is clear that some standalone private companies cannot manually manage tokens, that does not bring any value, or even threaten title rights. In this sense the public authorities have more trust, however at the same time generate regulatory constraints and bureaucracy. We also found that such solution like hashing of title (deed) records in the cadastral/land registries will not significantly improve relations in the domain.

As the result of this research it is developed the concept of a high-level design of the technology, capable to manage title rights on the blockchain which includes three-level mechanism of 1) e-voting, which provides for a democratic implementation of governing algorithms; 2) Smart Laws, as the concept of high level "smart" algorithms that implement (by e-voting) existing laws related to property rights in a form of the program/protocol; and 3) smart contract templates which are based on the smart laws, that allows people to manage their title rights online.

Acknowledgments

This paper is an outcome of the PhD research performed inside of the Joint International Doctoral (Ph.D.) Degree in Law, Science and Technology, coordinated by the University of Bologna, CIRSFID in cooperation with University of Turin, Universitat Autònoma de Barcelona, Tilburg University, Mykolas Romeris University, The University of Luxembourg. Thanks to my supervisor Prof. Marta Poblet Balcell, RMIT University (Melbourne, Australia), and Pompeu Casanovas, Universitat Autònoma de Barcelona (Barcelona, Spain).

References

1. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf.
2. Nachiappan, Crosby, M., Pattanayak, P., Verma, S., Kalyanaraman, V.: BlockChain Technology: Beyond Bitcoin. Appl. Innov. Rev. (2016).
3. Distributed Ledger Technology & Cybersecurity: Improving information security in the financial sector. (2016).
4. Ethereum Project, https://www.ethereum.org.
5. Nxt - The Blockchain Application Platform, https://nxt.org.
6. Ethereum Wiki, https://github.com/ethereum/wiki/wiki/Glossary.
7. Szabo, B.N., Practice, C., Controls, A., Interchange, E.D., Costs, C.T., Economics, T., Security, O., Enforceability, O., Phases, C., Intermediaries, A., Protocols, C., Cryptography, S.K., Computation, M.C., Authentication, P., Logs, P.T., Certificates, B., Transfer, U., Objects, C., Cash, D., Management, C.R.: Formalizing and Securing Relationships on Public Networks.
8. Velox.re, https://www.velox.re/.
9. Surda, P.: Economics of Bitcoin : is Bitcoin an alternative to at currencies and gold ?, http://nakamotoinstitute.org/static/docs/economics-of-bitcoin.pdf, (2012).

10. Mirkovic, J.: Blockchain Pilot Program. Final Report. (2017).
11. UBITQUITY - The First Blockchain-Secured Platform for Real Estate Recordkeeping.
12. Bitland. Land Title Protection Ghana, http://www.bitland.world/about/.
13. Real Estate Land Title Registration in Ghana Bitland, http://bitlandglobal.com/.
14. Blockchain and Future House Purchases, https://chromaway.com/landregistry/.
15. Flip Blog, https://blog.flip.lease/.
16. REX. The Global Real Estate Data Marketplace, http://rexmls.com/.
17. Chavez-Dreyfuss, G.: Ukraine launches big blockchain deal with tech firm Bitfury, http://www.reuters.com/article/us-ukraine-bitfury-blockchain-idUSKBN17F0N2.
18. Vavilov, V.: The Bitfury Group, http://bitfury.com/.
19. Xinyuan Real Estate Co., Ltd. Announces Blockchain-Powered Real Estate Finance Technology Platform, https://www.prnewswire.com/news-releases/xinyuan-real-estate-co-ltd-announces-blockchain-powered-real-estate-finance-technology-platform-300299818.html.
20. Antadze, L.: Bits of “Blockchain” is pointless in Govtech, https://medium.com/@lashaantadze/bits-of-blockchain-is-pointless-in-govtech-5b8044fd2d9c.
21. Sharma, T.: How Blockchain Can Be Used In Escrow & How It Works?, https://www.blockchain-council.org/bitcoin/blockchain-can-used-escrow-works/.
22. Buterin, V.: Ethereum and Oracles, https://blog.ethereum.org/2014/07/22/ethereum-and-oracles/.

Proof-Carrying Smart Contracts

Thomas Dickerson[1], Paul Gazzillo[2], Maurice Herlihy[1], Eric Koskinen[2], and Vikram Saraph[1]

[1] Brown University
[2] Stevens Institute of Technology

Abstract. This is preliminary work on reconciling the apparent contradiction between the immutability of idealized smart contracts and the real-world need to update contracts to fix bugs and oversights. Our proposed solution is to raise the contract's level of abstraction to guarantee a specification φ instead of a particular implementation of that specification. A combination of proof-carrying code and proof-aware consensus allows contract implementations to be updated as needed, but so as to guarantee that φ cannot be violated by any future upgrade.
We propose proof-carrying smart contracts (PCSCs), putting formal correctness proofs of smart contracts *on the chain*. Proofs of correctness for a contract can be checked efficiently by validators, who can enforce the restriction that no update can violate φ. We discuss architectural and formal challenges, and include an example of how our approach could address the well-known vulnerabilities in the ERC20 token standard.

The Scalability of Trustless Trust

Dominik Harz[1] and Magnus Boman[2,3]

[1] IC3RE, Imperial College London, SW7 2RH London, UK,
[2] RISE, Box 1263, SE-16429 Kista, Sweden,
[3] KTH/ICT/SCS, Electrum 229, SE-16440 Kista, Sweden

Abstract. Permission-less blockchains can realise trustless trust, albeit at the cost of limiting the complexity of computation tasks. To explain the implications for scalability, we have implemented a trust model for smart contracts, described as agents in an open multi-agent system. Agent intentions are not necessarily known and autonomous agents have to be able to make decisions under risk. The ramifications of these general conditions for scalability are analysed for Ethereum and then generalised to other current and future platforms.

Keywords: Trustless trust, smart contract, agent, Ethereum, blockchain, scalability, multi-agent system, distributed ledger

1 Introduction

Turing-complete programming languages allow creating a generic programmable blockchain by means of smart contracts [30]. A smart contract can be defined as a decentralised application executed on the distributed P2P network that constitutes the blockchain. The smart contract captures the formalisation of electronic commerce in code, to execute the terms of a contract. However, a smart contract is, in fact, neither smart nor a contract. In practice, it codes an agreement about what will come to pass, in the form of a production rule. Since there cannot be a breach of contract—which would happen only if one or more parties would not honour the agreement—thanks to how this production rule is coded, a smart contract is not a contract. Since there is no opportunity for learning on the contract's behalf, it is also not smart.

Smart contracts do code the preferences of their owners, and their negotiating partners as appropriate, with respect to the decision under risk or uncertainty. They react on events, have a specific state, are executed on a distributed ledger, and are able to interact with assets stored on the ledger [28]. Ethereum offers smart contracts through its blockchain. The Ethereum Virtual Machine (EVM) handles the states and computations of the protocol and can theoretically execute code of arbitrary algorithmic complexity [3]. Using Ethereum, developers can implement smart contracts as lines of code in an account that execute automatically when transactions or function calls are sent to that account. The outcome is final and agreed on by all participants and blockchains can thus enable a system of trust.

In Ethereum, smart contracts can interact through function calls via their Application Binary Interface (ABI). Single smart contracts or multiple smart contracts together can act as decentralised autonomous organisations by encoding the rules of interaction for the organisation's inner and outer relationships (e.g., The DAO, MakerDAO). *Full nodes* store the distributed ledger and validate new blocks in the chain *pro bono*. Permission-less blockchains limit the complexity of computation tasks and thus, the scalability of these blockchains. When utilising smart contracts, external services can be required to circumvent these computational limitations to code the preferences of their owners. The result of computations performed by external parties are not subject to the consensus protocol of the underlying blockchain, and their provided solution or correct execution cannot be formally verified. Hence, the oft-cited benefit of blockchains allowing for transparency over every transaction and enforced trust through a consensus mechanism cannot be guaranteed with external entities [17]. A trust model for smart contracts in permission-less blockchains is thus missing, a fact that limits their adaptability. Earlier trust models used in related applications, such as those devised for quantitative trading or speculative agent trading (see the patent text [14] for a good indication of this range), need to be adjusted for the inherent transparency and particular trust implications of blockchain systems. We propose a model that incorporate all these aspects.

2 Method

We answer the following research questions:

1. Which models of trust can be applied to smart contracts to reflect public permission-less blockchains?
2. What can be done to clarify the link between, on the one hand, the preferences and intentions of authors of smart contracts and, on the other hand, the run-time properties of those smart contracts?
3. How can properties of trust models be applied to verify computations in permission-less blockchains?

Question 1 is analysed in two steps. First, the applicability of agent-based trust models for smart contracts is evaluated by deducing their strong and weak notions based on agent theory. Second, a trust model suitable for smart contracts in permission-less blockchains is developed, based on a review of existing multi-agent system trust models [23]. Question 2 is analysed deductively, based on literature on decision theory and decision analysis, and on limitations of formal representations of preference, and their logical closure, e.g., what can be derived from them. Question 3 is investigated instrumentally, by developing an algorithm for verifiable computations. The development of the algorithm followed a deductive method of merging verifiable computation concepts using blockchains [34] [29] with cloud and distributed systems research [6] [7]. This revolves around preserving privacy of user data, whereby aspects of the blockchain are used to enforce the algorithm [34], and on verifiable computation for Ethereum using

computation services inside the blockchain [29]. In the latter, a verification algorithm with dispute resolution and an incentive layer were suggested, and the relevant assumptions critically assessed to develop a new algorithm, since their proposal had two practical issues: First, the verification game includes a 'jackpot' to reward solvers and verifiers for their work. This introduces an incentive to steal the jackpot by solvers and verifiers colluding to receive the jackpot without providing a correct solution. Second, they propose to implement the computation tasks in C, C++, or Rust code using the Lanai interpreter implemented as a smart contract on Ethereum. This limits the flexibility of computation services by forcing them to use one of the three programming languages. The objective of the here presented algorithm is to achieve:

1. execution of arbitrary computations requested from a smart contract in Ethereum, and executed outside the blockchain;
2. verification of the computation result achievable within reasonable time, i.e., $\mathcal{O}(n)$;
3. guarantees that the result of the computation is correct without having to trust the providing service.

Our development was experimental and explorative. Different parameters and the agents they pertain to were first considered in a pen and paper exercise, then validated via qualitative assessment as well as quantitative analysis. The quantitative experiments constitute an evaluation basis for the last two algorithm objectives.

3 Explicating Smart Contracts

Consensus protocols are used to decide upon the state of the distributed ledger [21]. This ledger is in permission-less blockchains accessible to anyone participating in the network and through blockchain explorers even to entities outside of the network. This means everyone is able to see for example which public key owns the most Ether. Also, each transaction can be inspected, making it possible for participating parties to monitor the progress of their transaction. To provide an incentive to the miner and prevent unnecessary changes to the ledger, blockchains introduce fees on executing transactions [21]. In Ethereum, the blockchain stores transactions and the code of smart contracts as wellas their state. Hence, the state of a smart contract needs to be updated in the same fashion as executing a transaction including fees, consensus, and mining time.

Smart contracts on Ethereum are executed by each node participating in the P2P network and hence operations are restricted to protect the network [31]. To circumvent operational issues (e.g., someone executing a denial of service attack on the network), Ethereum introduces a concept to make users pay for execution of a smart contract functions, and the EVM supports only certain defined operations [31], with each operation coming with a certain cost referred to as *gas*. Before executing a state-changing function or a transaction, the user

has to send a certain amount of gas to the function or the transaction. Only if the provided amount of gas is sufficient for the function or transaction to execute, it will successfully terminate. Otherwise, the transaction or function will terminate prematurely, with results contingent on the handling of the smart contract function.

We now look at two ways of explicating the roles that smart contracts may take on. First, the agent metaphor is employed to provide an informal understanding in terms of a widely accepted and understood terminology. Second, the concept of utility is employed to provide a formal understanding of how the preferences and intentions of smart contract owners may be encoded in the contract itself.

3.1 Smart Contracts as Agent Systems

Agents have certain properties separable in weak and strong notions [32]. Weak notions include *autonomy*, *pro-activeness*, *reactivity*, and *social ability*. *Autonomy* refers to the smart contract ability to operate without a direct intervention of others and include control over their actions and state. In Ethereum, the state of smart contracts is maintained on the blockchain, while the actions are coded into the smart contract itself. These actions can depend on the state, thus providing a weak form of autonomy. *Pro-activeness* describes goal-directed behaviour by agents taking initiative. This is somewhat limited in Ethereum, as smart contracts currently act on incoming transactions or calls to their functions. However, if one perceives an agent as a collection of multiple different parts, smart contracts might well be extended by external programs triggering such initiatives. Thereby, the limitations set by Ethereum can be circumvented and an agent with pro-active notions can be created. The result is in effect a multi-agent system and can be analyzed as such. *Reactivity* is based on perception of an agent's environment and a timely response to those changes. By design, smart contracts only have access to the state of the blockchain they are operating in. Reactivity for state changes in Ethereum is reached via event, transaction, or function implementation. To react to environment changes outside of the blockchain (e.g. executing a function based on changes in stock market prices) requires importing this information to the blockchain via e.g. Oracles [4]. *Social ability* enables the potential interaction with other agents or humans through a communication language. In Ethereum, users and contracts are identifiable by their public key [31] and interaction is possible through transactions or function calls on smart contracts.

Strong notions include properties such as *beliefs and intentions*, *veracity*, *benevolence*, *rationality*, and *mobility*. As mentioned in the introduction above, pro-activeness is somewhat limited in Ethereum smart contracts, and so these properties are present only to a limited extent. The two properties *veracity*, which refers to not knowingly communicating false information, and *rationality*, describing the alignment of the agent's actions to its preferences, both pertain to the incentives an author of a smart contract might have to develop an agent which is rational but not truthful, in order to maximise profits. This can be

deliberate so that the agent correctly encodes the true preferences of the smart contract owner, or non-deliberate, in which case the owner preferences might be inadequately coded. To deal with the uncertainty of agent intentions, three approaches have emerged. First, security approaches utilise cryptographic measures to guarantee basic properties such as authenticity, integrity, identities, and privacy [23]. Within blockchains, this is mainly achieved through cryptographic measures, which do not provide trust in the content of the messages. Second, institutional approaches enforce behaviour through a centralised authority. This entity controls agents' actions and can penalise undesired behaviour. Governance functions enforcing behaviour not defined in the core protocol do not exist. Third, social approaches utilise reputation and trust mechanisms to e.g. select partners, punish undesired behaviour, or evaluate different strategies. In blockchains, there is no system of trust implemented in the core protocol, which would rate behaviour according to certain standards. These three approaches are complementary and can be used to create a system of trust [23]. Trust research and current implementations are primarily focused on the first two approaches. This allows creating agents on a platform that enforces these defined trust measurements [1] [26] [24] [20].

3.2 Utility and Risk

Some researchers believe that all game-theoretical aspects of making decisions can be pinned down by logical axiomatizations: it is only a matter of finding the right axioms. Game-theoretical studies often concentrate on two-person games, one reason being that many conflicts involve only two protagonists. In any game, the players may or may not be allowed to cooperate to mutual advantage. If cooperation is allowed, the generalized theory of n-person games can sometimes be reduced to the one for two-person games, since any group of cooperating players may be seen as opposing the coalition of the other players. In the case of smart contracts, this would allow for an owner of multiple contracts (in effect, a multi-agent system) to maximize the utility of interplaying contracts by employing game theory, at least on paper. For a given set of smart contracts, the problem is how to determine a rule that specifies what actions would have been optimal for the smart contract owner. Actions could here pertain to details of a particular contract, or to the order of their execution, for instance. Comparing different rules measures the risk involved in consistently applying a particular rule, e.g., a chain of smart contract employment. Formally, we wish to determine a decision function that minimizes this risk. The simpler case of handling risk is in decisions under certainty. This means that the owner of one or more smart contracts can predict the consequences of employing them. This represents the ideal case in which all smart contracts execute as intended. Thus, the owner simply chooses the alternative whose one and only possible consequence has a value not less than the value of any other alternative. This seems simple enough, but it is necessary to investigate a bit further what the value of a consequence denotes. The preferences of the owner should be compatible with the following axioms (A is not preferred to B is henceforth denoted by $A \leq B$).

$\leq$ is a weak ordering on the set of preferences P:
A1. (i) Transitivity: If $A \leq B$ and $B \leq C$, then $A \leq C$, for all A, B, and C in P.
A1. (ii) Comparability: $A \leq B$ or $B \leq A$, for all A and B in P.
From this, we may derive the relation of indifference and strict preference, and we state the consistency criteria for these:
A2. (i) $A = B$ is equivalent to $A \leq B$ and $B \leq A$, for all A and B in P.
A2. (ii) $A \leq B$ is equivalent to $A \leq B$ and not $B \leq A$, for all A and B in P.
However, A1 implies that the owner has to admit to all consequences being comparable. This is typically not the case in smart contracts, and it becomes necessary to replace Comparability with Reflexivity, yielding a partial ordering instead:
A1. (iii) Reflexivity: $A \leq A$, for all A in P.

There is much to be gained by representing the preference ordering as a real-valued order-preserving function. If we cannot find such a function there is not much sense in speaking of the numerical value of a sequence of employed smart contracts, and we might as well throw a coin for deciding. Assuming axioms A1 and A2 hold, we must find a function $f(X)$ with the property $f(A) \leq f(B)$ iff $A \leq B$, which we can always do fairly easily for decisions under certainty [13], but we now turn to decisions under risk, which is the class of decisions that normally pertain to owners of smart contracts. In the Bayesian case, with subjective probabilities, we can think of a smart contract employment S as consisting of a matrix of probabilities $p_1, ..., p_n$ and their corresponding consequences $c_1, ..., c_n$. Then the real-valued function $f(X)$ we seek lets us compute the value of S as $\Sigma p_i f(c_i)$. This fixes one possible definition of an agent as rational, by making it maximize its own utility (in accordance with its preferences, i.e. with the preferences it codes). Formally, an agent accepts the utility principle iff it assigns the value $\Sigma p_i v_i$ to S, given that it has assigned the value v_i to c_i. Any ordering Ω of the alternatives is compatible to the principle of maximizing the expected utility iff $a \Omega b$ implies that the expected value of a is higher than the expected value of b. In other words, we are now free to start experimenting with various axiom systems for governing the owners, or at least recommending them actions based on the smart contracts they have at hand. While game-theoretic axiom systems have been favoured among agent researchers, a wide variety of axiomatizations are surveyed in the more formal literature [12] [19].

4 A Trust Model for Smart Contracts

From the 25 models covered in [23], five consider global visibility and nine consider cheaters. The overlap of those models leaves one model focusing on reputation of actors in electronic markets [25]. The core idea is to use incentives to encourage truthful behaviour of agents in the system by social control. Social control implies that actors in the network are responsible for enforcing secure interactions instead of using an external or global authority.

Assuming a rational agent, there is a possible motivation to break protocol if this maximizes utility. Speculation-free protocols have been recommended for

some agent applications, but the Ethereum smart contract environment is much too complex to allow for such control features, which require equilibrium markets [27]. To provide a certain level of trust, new agents have to deposit a certain cryptocurrency value for participation, and this deposit is returned when an agent decides to stop participating. However, dishonest or corrupt agents can be penalised by either destroying their deposit or distributing it to honest agents. This is in line with norm-regulation of agent systems [2] and does not make any other strong requirements on models. Norm-regulation has been formalized for multi-agent systems, e.g., in the form of algebra [22].

Gossiping can be used to communicate experiences with other agents in a P2P fashion and thereby establish trust or reputation. In the protocol of Bitcoin or Ethereum gossiping is the basis for propagating new transactions and subsequently validating blocks [10]. A similar approach can be taken for smart contracts, whereby agents could exchange knowledge or experiences of other agents [8]. Reputation of an agent is based on its interaction with other agents, whereby agents mutually need to sign a transaction if they are satisfied with the interaction. Over time, an agent collects these signed transactions to build up its reputation. However, this model is prone to colluding agents boosting their reputation [5]. Trust can also be implemented by relying on independent review agents [15] [16] [9]. However, both gossiping and review agents are subject to detection rate issues.

5 Applying Trust Measures to Verifiable Computation

Due to the restrictions set by the EVM (i.e. gas cost of operations), implementing functions in Ethereum with a complexity greater than $\mathcal{O}(n)$ is not feasible. To circumvent these limitations, computations can be executed outside of Ethereum and results stored on the blockchain. We present an algorithm to achieve verifiable computations outside of Ethereum through measures presented in the trust model. Agents' rational behaviour can be aligned to the overall objective of the algorithm. The actors involved in the verifying computation algorithm are presented in Fig.1. *Users* request solving a specific computation problem. They provide an incentive for solving and verifying the problem. *Computation services* provide computation power in exchange for receiving a compensation. For participation, they are providing a deposit. One of the computation services acts as a *solver* and at least one other computation service acts as a *verifier*. *Judges* decide whether basic mathematical operations are correct or not. They are neutral parties and are not receiving any incentives. An *arbiter* enforces the verifiable computation algorithm when users request a new computation.

Users are assumed as agents with the objective to receive a correct computation. They are required to send a fee to reward solvers and verifiers for executing the computation. This fee depends on the complexity of the computation to be performed, the complexity of the input data, and the number of verifiers. Computation services are assumed to optimise their incentive. They might purposely communicate false information to maximise their incentive. Fur-

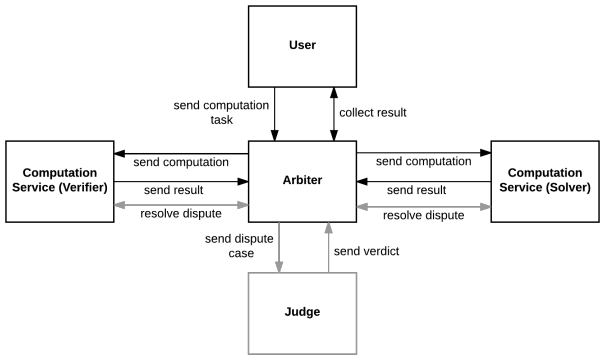


Fig. 1: Overview of actors in the verification algorithm.

ther, enough computation services are available (i.e. a minimum of 2) to execute the computation with at least one verifier. The probability of detecting a false computation depends on the number of verifiers in the algorithm. The arbiter and judge are trusted by participating parties, respectively enforcing the algorithm and reaching a verdict. This is a strong assumption in a trustless system and needs to be justified. To limit their incentive for undesired behaviour (i.e. cheating) in the algorithm, these two agents are not rewarded for taking part in the computations. Thus, their work is *pro bono* and only the operational cost in gas are covered.

Alternatively and not further covered in this paper, other approaches limit or eliminate trust in arbiter and judge. First, following the trust is risk approach [18], a network of trusted entities with a fixed amount of deposited value could be created to find arbiters and judges trusted commonly between computation services and users. Second, a user might create their own arbiter and judge, while storing the fee in an escrow contract between user and computation services. The computation services store an encrypted hash of the result in the escrow contract. Upon completion of the protocol, the user issues the payment and receives the result in full. Third, the protocol could be executed with different test cases while results would be publicly stored on the blockchain. Thus, a user and computation service could verify correct execution of the protocol, if arbiter and judge remain unchanged.

5.1 Algorithm

The algorithm is initiated when a user requests a computation by sending the input data, the operation to be performed, and the desired number of verifiers to the arbiter. One computation service is randomly determined as a solver, and the other(s) are randomly assigned as verifiers by the arbiter. The user instructs the arbiter to forward the input data and operation to the computation services smart contracts, triggering the off-chain computation by sending a request

through an oracle. This requires sending a fee for the computation as well as providing the fee for using the oracle. Verifiers and the solver report their result back to the arbiter. If all results are reported back, then the user can trigger the arbiter to compare the available results. If the solver and all participating verifiers agree on one solution, the algorithm is finished and the user can collect the result. However, if at least one verifier disagrees with the solver the user can initiate a dispute resolution algorithm. The dispute resolution is inspired by a technique introduced in [7], [6], and [29]: to split up the operation into simple parts with intermediary results until the computation is simple enough for the judge to solve it. Overall and intermediary results are stored in a Merkle tree for the solver, and each verifier challenging the solver. The comparison is achieved through a binary search on the trees. The root of the tree encodes the overall result, while the leaves in the lowest layer encode the input data. Leaves in between represent intermediary results.

5.2 Interactions

Under the assumption that arbiter, judge, and user behave rational and follow the algorithm, computation services have a combination of four different behaviours with respect to their role as solver S or verifier V. The behaviours are summarised in Table 1 with either verifiers accepting the solution (i.e. V_A) or challenging the solution (i.e. V_C). S profits the most if it provides a correct solution, which is challenged by V, while V profits the most when S provides a false solution and V is able to challenge it. The problematic case is that the incentives for accepting a false or correct solution are the same. To prevent this from happening we will consider the behaviour of V and S in detail.

Table 1: Possible behaviours of computation services as solver S and verifier V, whereby all verifiers behave the same.

		S	
		correct solution	false solution
V	challenge	S receives S fee share	S receives nothing
		S receives V_C fee share	V_C receives V_C fee share
		V_C receives nothing	V_C receives S fee share
	accept	S receives S fee share	S receives S fee share
		V_A receives V_A fee share	V_A receivesV_A fee share

Case 1: S provides a correct solution and no V challenges the solution. Agents behave as intended by the algorithm. As no V challenges the solution, the judge is not triggered and the fee is equally split between S and the involved V.

Case 2: S provides a correct solution and at least one V challenges the solution. This is an undesired behaviour since the solution provided is actually

correct. This triggers the dispute resolution with a verdict by the judge determining S as correct. In this case S profits from the extra work due to the additional dispute steps by receiving the fee share of V_C. V_A receive their part of the fee since their amount of work remained the same.

Case 3: S provides a false solution and no V challenges the solution. S and all V would receive their share of the fee. This is an undesired behaviour in the algorithm as it would flag a false result as correct. To prevent this from happening two measures are used. First, computation services do not know their role in advance as they are randomly assigned by the arbiter. If several services collude to provide false solutions, all of them would need to work together to provide the "same wrong" result. However, if just one V_C exists, it profits by gaining the fee shares of itself, S, and all V_A. Thus, second, the user is able to determine the number of V for each computation. The probability of having at least one V_C depends on the prior probability p of V providing correct or false solutions and the number n of V in the computation.

Case 4: S provides a false solution and at least one V_C challenges the solution. Hereby, S and V_A are not receiving their share of the fee, which goes to all V_C. This is based on the verdict by the judge. However, this is also an undesired case since the user does not receive a solution to his computation.

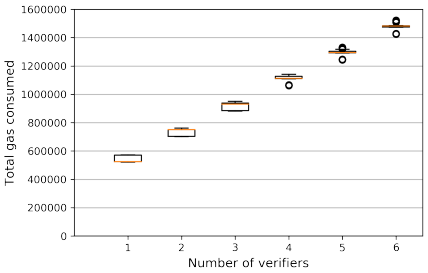
Considering the four scenarios, rational S is trying to receive its share of the incentive and get a chance to receive fees of any V challenging a correct solution. The strategy for S considering V is to provide a correct solution to the problem. V profits the most form challenging a false solution. A rational V provides the correct solution to a computation to receive its fee share or to have the chance of becoming a challenger to a false solution. Arguably, S and V could try to deliver a false solution to save up on computation cost or trick the user. In this case, the probability of discovering the false solution relies on the number of Vs and the prior probability of cheating Vs. If a V delivers a false solution, it must be the same solution as S' to not trigger the dispute resolution. Moreover, by destroying the services' deposits and excluding them from the algorithm after detected cheating, the prior probability of having such a service can be reduced.

5.3 Implementation and experiments

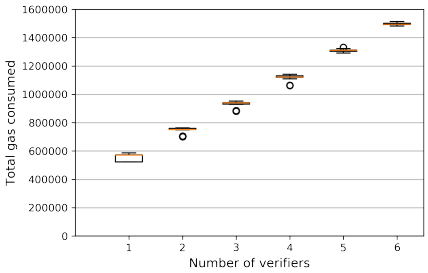
The algorithm was implemented using Solidity smart contracts and AWS Lambda external computation services. The quantitative analysis is conducted by executing experiments with one exemplary type of computation. The computation is a multiplication of two integers to simplify the verification steps in the algorithm. The results depend on external and internal parameters of the algorithm. Externally, the prior probability of computation services providing false solutions is considered. Internally, the number of verifiers the user requests for each computation are examined. Experiments are executed for each different configuration of parameters to determine gas consumption and outcome of the computation. Assuming a potentially large number of computation services ($> 10,000$), this gives a confidence level of 95% and a maximum confidence interval of 3.1 for the three different prior probabilities. Before each iteration of the experiment, the

environment is initialised with a new set of smart contracts. Experiments are executed within *TestRPC* [11].

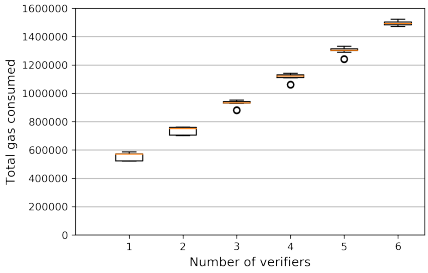
Reporting the amount of gas used equals the time and space complexity of the algorithm, as gas consumption is determined by the type and number of operations in the EVM. It further excludes the time used for sending transactions or calls. Independent of the prior probability of false solutions, the μ gas consumption increases linearly as presented in Fig.2. Further, σ decreases with an increasing number of verifiers. At a low number of verifiers, the dispute resolution is less likely triggered, leading to a higher σ in gas consumption. With an increasing number of verifiers, the probability of triggering the dispute resolution increases. As the dispute resolution is almost always triggered, σ is reduced.



(a) 30% of computation services providing incorrect solutions.



(b) 50% of computation services providing incorrect solutions.



(c) 70% of computation services providing incorrect solutions.

Fig. 2: Total amount of gas used by algorithm with different number of verifiers and percentage of computation services providing incorrect solutions. Each combination of specific number of verifier(s) and percentage of computation services with incorrect solutions with $N = 1000$.

The algorithm is tested for three different cases of verification: First, the algorithm can accept a correct solution. Second, each verifier agrees with the solver although the solution is not correct. The dispute resolution is not triggered and the user receives a false solution marked as correct. Third, at least one

verifier disagrees with the solver providing a false solution and the judge rules that the solver's solution is false. For the second case, invoking the dispute resolution depends on the prior probability of computation services providing false solutions described by $P(V_C) = 1 - p^n$. The experiments as shown in Table 2 indicate that the expected and actual value are similar for $p = 0.5$. However, for $p = 0.3$ and $p = 0.7$ the actual values are below the expected ones. Since the experiment is executed with a confidence level of 95% and interval of 3.1, those changes are accounted towards sampling size not being a perfect representative of the actual distribution. Also, the random assignment of false and correct computation services could be a cause for having a higher detection rate.

Table 2: Comparison of expected and actual probabilities of accepting a false solution in the algorithm.

Prior p	**Verifiers** n	**Expected false** [%]	**Actual false** [%]
0.3	1	9.0	2.7
0.3	2	2.7	0.0
0.3	3	0.81	0.0
0.3	4	0.243	0.0
0.3	5	0.0729	0.0
0.3	6	0.02187	0.0
0.5	1	25.0	28.6
0.5	2	12.5	12.2
0.5	3	6.25	4.6
0.5	4	3.125	1.2
0.5	5	1.5625	0.0
0.5	6	0.78125	0.0
0.7	1	49.0	41.2
0.7	2	34.3	24.4
0.7	3	24.01	12.1
0.7	4	16.807	4.9
0.7	5	11.7649	2.9
0.7	6	8.23543	0.0

6 Discussion

Within the presented trust model, deposits are simple to implement in permissionless blockchains that already have a cryptocurrency. However, the deposit value can be volatile. This poses two risks: Either the escrow or independent entity maintaining the deposit may be motivated to steal the deposits, or the deposit value might be so little that its trust-building attribute vanishes. To prevent this, the deposit value could be bound to a fiat currency or a stable asset. The deposit can also be dynamically adjusted and deposits only kept a short time or one iteration of interactions. Gossiping could be used as a basis to communicate

experiences with other agents. In permission-less blockchains, the agents can use a common protocol to exchange this information and use a rating approach [33]. Yet, gossiping can be misused by agents to boost their own reputations by executing Sybil attacks. Review agents can be used that reach a verdict on a specific issue or problem. Their implementation is simple and potential scenarios to manipulate agents' reputations are prevented. However, the judge or review agent needs to be trusted by other agents. The algorithm is based on its actors and their interaction. The idea of arbiter, judge, user, and computation services is strongly influenced by [29] and [34]. The main differences are in the idea of using a jackpot to reward verifiers as well as the implementation either entirely on Ethereum or using external computation services. Moreover, the algorithm defers from [34] as its goal is to deliver verifiable computations for entities (i.e. users or smart contracts) on the blockchain, while [34] primarily delivers privacy-preserving computations, where blockchain enables the algorithm.

The algorithm cannot guarantee to detect false solutions. It is based on the assumption that solvers and verifiers behave as desired (i.e. delivering correct solutions), as their strategy is aligned with the incentives provided by the algorithm. This assumption is based on game-theoretic properties. The algorithm leaves no dominant strategy considering the interactions in Table 1. S can choose either to provide a correct or false solution and V can challenge or accept. Only when considering both agents, a Nash equilibrium exists. If there is a (high) probability that a V_C exists, the only valid strategy for S is to provide a correct solution. Consequently, V in turn has to provide a correct solution, which accepts correct S and challenges false S. In the algorithm, both S and V providing correct solutions gives a Pareto efficient result. If they change their strategy under the assumption that no V_C exists, their utility remains the same. However, a V has an incentive to challenge a false solution, which would increase his utility and reduce the utility of the others. Social welfare considers the sum of all agent's utilities depending on their strategy which can be disregarded in permission-less blockchains since overall the agent wants to optimise his utility independent of the overall utility. Specifically, the overall utility is potentially unknown to an individual agent, since he is unable to determine with certainty the utility of other agents.

7 Conclusion

On permission-less blockchains like Ethereum, rational agents through smart contracts code the preferences of their owners. This could motivate maximizing their utility by dishonest behaviour, and hence, further social control mechanisms are required. We have presented a trust model for smart contracts in permission-less blockchains that incorporate state-of-the-art research into deposits, reputation, and review agents for social control. Trust can be extended to entities outside of permission-less blockchains through applying the trust measures presented in our model. An example application is an algorithm implementing verifiable computation. The model includes users requesting computational

tasks, computational services providing solutions and acting either as solver or verifier, arbiters enforcing the algorithm, and judges resolving disputes. Due to the incentive structure and the potential penalty cause by cheating, providing correct solutions to the computation task is a Nash equilibrium. Under the assumption that arbiter and judge are trusted, the algorithm detects false solutions provided based on a probability distribution. The algorithm is realised as Solidity smart contracts and AWS Lambda functions, implementing verification of multiplying two integers. Experiments show that with six verifiers the algorithm detects cheaters with prior probabilities of 30%, 50%, and 70% dishonest computation services. Experiments show that the algorithm performs overall with a linear time and space complexity depending on the number of verifiers.

As future work, we leave eliminating trust requirements regarding arbiter and judge by a fully decentralised algorithm.

Acknowledgement The authors thank Babak Sadighi and Erik Rissanen for comments and discussions, Daniel Gillblad for important support for Magnus Boman's part of the project. Also, the authors thank Outlier Ventures Ltd. for partly funding Dominik Harz' share of the project.

References

1. Balakrishnan, V., Majd, E.: A Comparative Analysis of Trust Models for Multi-Agent Systems. Lecture Notes on Software Engineering 1(2), 183–185 (2013)
2. Boman, M.: Norms in artificial decision making. Artificial Intelligence and Law 7(1), 17–35 (1999)
3. Buterin, V.: A Next-Generation Smart Contract and Decentralized Application Platform (2013), https://github.com/ethereum/wiki/wiki/White-Paper
4. Buterin, V.: Chain Interoperability. Tech. Rep. 1, R3CEV (2016)
5. Can, A.B., Bhargava, B.: SORT: A Self-ORganizing Trust Model for Peer-to-Peer Systems. IEEE Transactions on Dependable and Secure Computing 10(1), 14–27 (2013)
6. Canetti, R., Riva, B., Rothblum, G.N.: Practical delegation of computation using multiple servers. In: Proceedings of the 18th ACM conference on Computer and communications security - CCS '11. p. 445. ACM Press, New York, New York, USA (2011)
7. Canetti, R., Riva, B., Rothblum, G.N.: Refereed delegation of computation. Information and Computation 226, 16–36 (2013)
8. Carboni, D.: Feedback based Reputation on top of the Bitcoin Blockchain (2015)
9. Cerutti, F., Toniolo, A., Oren, N., Norman, T.J.: Context-dependent Trust Decisions with Subjective Logic (2013)
10. Decker, C., Wattenhofer, R.: A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In: Pelc, A., Schwarzmann, A.A. (eds.) Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9212, pp. 3–18. Springer International Publishing, Cham (2015)
11. Ethereum: Ethereum TestRPC (2017), https://github.com/ethereumjs/testrpc

12. Fishburn, P.: Foundations of Decision Analysis: Along the way. Management Science 35, 387–405 (1989)
13. French, S. (ed.): Decision Theory: An Introduction to the Mathematics of Rationality. Halsted Press, New York, NY, USA (1986)
14. Hoffberg, S.: Multifactorial optimization system and method (Apr 19 2007), https://www.google.com/patents/US20070087756, uS Patent App. 11/467,931
15. Huynh, T.D., Jennings, N.R., Shadbolt, N.R.: An integrated trust and reputation model for open multi-agent systems. Autonomous Agents and Multi-Agent Systems 13(2), 119–154 (2006)
16. Jakubowski, M., Venkatesan, R., Yacobi, Y.: Quantifying Trust (2010)
17. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In: 2016 IEEE Symposium on Security and Privacy (SP). vol. 2015, pp. 839–858. IEEE (2016)
18. Litos, O.S.T., Zindros, D.: Trust Is Risk: A Decentralized Financial Trust Platform. IACR Cryptology ePrint Archive 2017, 156 (2017)
19. Malmnäs, P.E.: Axiomatic Justifications of the Utility Principle. Synthese 99(2) (1994)
20. Mui, L., Mohtashemi, M., Halberstadt, A.: A computational model of trust and reputation. In: HICSS. Proceedings of the 35th Annual Hawaii International Conference on System Sciences. vol. 5, pp. 2431–2439. IEEE (2002)
21. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies - Draft. Princeton University Press, Princeton, NJ, USA (2016)
22. Odelstad, J., Boman, M.: Algebras for agent norm-regulation. Annals of Mathematics and Artificial Intelligence 42(1), 141–166 (2004)
23. Pinyol, I., Sabater-Mir, J.: Computational trust and reputation models for open multi-agent systems: A review. Artificial Intelligence Review 40(1), 1–25 (2013)
24. Ramchurn, S.D., Huynh, D., Jennings, N.R.: Trust in multi-agent systems. The Knowledge Engineering Review 19(01), 1–25 (2004)
25. Rasmusson, L., Jansson, S.: Simulated social control for secure Internet commerce. Proceedings of the 1996 workshop on New security paradigms - NSPW '96 pp. 18–25 (1996)
26. Sabater, J., Sierra, C.: Review on computational trust and reputation models. Artificial Intelligence Review 24(1), 33–60 (2005)
27. Sandholm, T., Ygge, F.: On the Gains and Losses of Speculation in Equilibrium Markets. In: Proceedings IJCAI'97. pp. 632–638. AAAI Press (1997)
28. Szabo, N.: Formalizing and Securing Relationships on Public Networks. (1997), http://ojphi.org/ojs/index.php/fm/article/view/548/469
29. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains (2017)
30. Vukolić, M.: Hyperledger fabric: towards scalable blockchain for business. Tech. Rep. Trust in Digital Life 2016, IBM Research (2016), https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf
31. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper pp. 1–32 (2014)
32. Wooldridge, M.: An Introduction to MultiAgent Systems. Wiley Publishing, 2nd edn. (2009)
33. Zhou, R., Hwang, K., Cai, M.: GossipTrust for Fast Reputation Aggregation in Peer-to-Peer Networks. IEEE Transactions on Knowledge and Data Engineering 20(9), 1282–1295 (2008)
34. Zyskind, G.: Efficient Secure Computation Enabled by Blockchain Technology. Master thesis, Massachusetts Insitute of Technology (2016)

Toward Cryptocurrency Lending

Mildred Chidinma Okoye[1,2] and Jeremy Clark[1]

[1] Concordia University
[2] Deloitte

Abstract. Lending has been posited as an application of blockchain technology but it has seen little real deployment. In this paper, we discuss the roadblocks preventing the effortless lending of cryptocurrencies, and we survey a number of possible paths forward. We then provide a novel system, Ụgwọ, consisting of experimental smart contracts written in Solidity and deployed on Ethereum to demonstrate how a decentralized lending infrastructure might be constructed.

1 Introductory Remarks

Lending has been posited as an application of blockchain technology but we have seen little real deployment of lending. In Section 2, we discuss roadblocks and possible paths forward. We do this in service of other researchers who might want to look at this issue—we view our own contributions as an initial look and not the final word in this complex area. We outline our agenda in a few steps: (1) we review the role of lending in a modern economy, (2) we identify the key tensions between cryptocurrencies like Bitcoin and Ethereum and lending, (3) we review proposals for lending, and (4) we suggest how to move forward. In Section 3, we present our lending infrastructure Ụgwọ which incorporates the points we discuss. Ụgwọ is designed to be flexible and extensible; traditional fiat-based lending is not one-size-fits-all and consists of a patchwork of loan structures, instruments, and intermediaries. We show some basic types of loans and basic types of risk mitigation as examples of what could be added to Ụgwọ to support an infrastructure for lending.

2 A Research Agenda for Cryptocurrency Lending

2.1 The role of lending in a modern economy

It is difficult to overstate the role of lending in a modern economy. Take, as an illustrative example, the role of a central bank; one of the main national institutes (along with the treasury) that cryptocurrencies aim to displace. First and foremost, a central bank is an actual bank, providing accounts for its member banks to deposit money and earn interest. Member banks provide interest-earning accounts to the public. Interest is paid to the public because banks use the deposited money to form loans. Because central bank interest rates are low, banks

prefer to lend to other banks any excess cash they hold at day's end instead of depositing them (other banks borrow to meet liquidity requirements). These loans earn interest, and central banks target this specific lending rate when they intervene in the economy. The most common intervention is the buying (circulating new money) or selling (removing circulating money) of government bonds, which are interest-earning loans from investors to the government. Central banks will also provide loans (of 'last resort') to banks unable to secure loans from other banks, typically during some sort of liquidity crisis. An economy without loans would have no interest rates, no bonds, and essentially nothing for a modern central bank to do.

2.2 Two critical issues for lending with cryptocurrencies

The crypto-economy is effectively an economy without loans. We identify two primary roadblocks:

- **Monetary instability.** While a loan might be in anything of value, it is typically done with money. Cash loans work best when the value of the money is relatively stable. By contrast, cryptocurrencies have historically appreciated in value over time (as of the time of writing). In a lending situation, this means the cash taker will end up owing far more than he borrowed. If the scenario were reversed and the currency depreciated rapidly, the cash provider would prefer to spend the money rather than locking it up in a loan where it will shed value over time. Even without long-term upward or downward drifts in value, short-term volatility adds risk to a loan for both the cash taker and the cash provider.
- **Counter-party risk.** While the hype surrounding blockchain technology centers on how it can enable trustless financial systems, there is no way to blockchain your way out of counter-party risk. If Alice truly lends money to Bob—truly in the sense that Bob fully owns it and can do with it as he pleases—then Bob can abscond with the money.

2.3 Existing proposals

A number of companies have launched loan products or systems based on cryptocurrencies. In the most common architecture, a central company arranges loans and the loans are simply denominated in cryptocurrencies like Bitcoin. These services vary from at interest bearing accounts to peer-to-peer lending for investment purposes to social justice orientations like mirco-lending for the unbanked or the subprime market. As opposed to our system Ụgwọ, these do use smart contracts to structure the actual loans.

2.4 Dealing with monetary instability

We summarize a few suggestions for adding stability to cryptocurrencies.

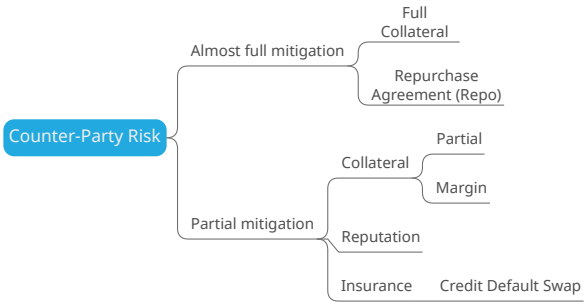


Fig. 1. Standard approaches to dealing with counter-party risk.

- The rate of release of new currency into the system could be modified to enable new currency to be introduced at (i) a more insightful rate or (ii) based on some internal metrics of the system like number of transactions. [*Remark:* an insightful rate has been elusive despite many alt-coins customizing the schedule and it is difficult to see how metrics could not be gamed].
- A cryptocurrency can also use explicit pegging but it is no better suited to this system than standard currencies.
- A central bank could manage currency circulation while allowing other aspects to be decentralized [4]. [*Remark:* Central banks have been historically unsuccessful at using money circulation as a target [7]].
- The loan could be use the cryptocurrency as the medium of exchange but use a stable (*e.g.,* government) currency as the unit of account.

In Ụgwọ, we use the last approach. In other words, a loan could be $100 USD paid in Ether at the exchange rate at loan time and repaid 3 months later at $110 USD paid in Ether at the new exchange rate. This approach requires the smart contract to be aware of the exchange rate which introduces a trusted third party, called an oracle [11] and is discussed further in the next section.

2.5 Dealing with counter-party risk

In Figure 1, we outline the basic approaches from finance for dealing with counter-party risk.

- *Full Collateral:* It is common for Bitcoin-based solutions, *e.g.,* for fair exchange [1,3,10] or payment channels [5,9], to deal with counter-party risk by requiring full collateral. This is a simple approach but one unlikely to scale to an entire economy: economic actors are chagrinned to leave money where it earns no interest and economic benefit.
- *Repurchase Agreement:* A loan collateralized fully with same currency as the loan is not a loan therefore collateral only works if it is something different of

the same value. If this something is on-blockchain (say a token representing something of value), the cash provider can have the collateral sit locked up in escrow (where it benefits neither the cash provider or taker) or could take full ownership of the collateral with the promise of returning it when the loan is repaid. This is a repurchase agreement and is common when the cash provider is perceived to be at less risk of absconding than the cash taker.

- *Partial Collateral:* The cash taker might stake something of lesser value than the loan in collateral, a third party to a loan might use partial collateral to insure a loan (see below), or sometimes loans are internal to a system such as leveraged positions in financial markets where the manager can liquidate the loan if the partial collateral (margin) dissipates due to market conditions.
- *Reputation:* A more abstract form of collateral is one's reputation and lending history. The difficulty with reputation is that it requires strong identities, something missing from decentralized currencies, as rogue entities can regenerate a new identity if the reputation of their old identity suffers and they can generate fake histories by lending to themselves with fake identities. These are not impossible to address but are difficulties.
- *Insurance:* Consider the case where Alice lends to Bob and does not trust him. If Alice trusts Carol and Carol trusts Bob, then Carol could insure the loan. Of course, Carol in this case could also just lend the money to Bob but there are a few scenarios where she might let Alice lend the money. One is if Carol's assets are not liquid. A second is that Carol might employ partial collateral: she could insure 100 loans of similar value but only stake 10% of the lent money as a margin against defaults. This costs her less than making the loans herself, and provides the cash providers insurance assuming the default rate is less than 10%. One standard financial instrument to implement this type of insurance, with some additional complexities discussed later, is a credit default swap (CDS).

3 The Ụgwọ Lending Infrastructure

Ụgwọ is an extensible system of smart contracts to enable different types of lending on Ethereum.[3] It is centred around recording credit events—when a party fails to fulfill the terms written in a loan contract—in a common ledger called a Credit Event object. We considered two implementation approaches:

- *Internal Variable.* In one approach, a loan has a credit event object within itself where the credit event is a variable contained within the loan contract. The issue with this approach is one of encapsulation: any external contract protecting the loan (via insurance or collateral) would have to reach inside the loan object when all it needs to know to function is whether a credit event occurred or not.
- *Object Oriented Approach.* It would be interesting if the Credit Event object sat at its own address such that protection contracts could be externally

[3] https://github.com/MildredOkoye/Ugwo

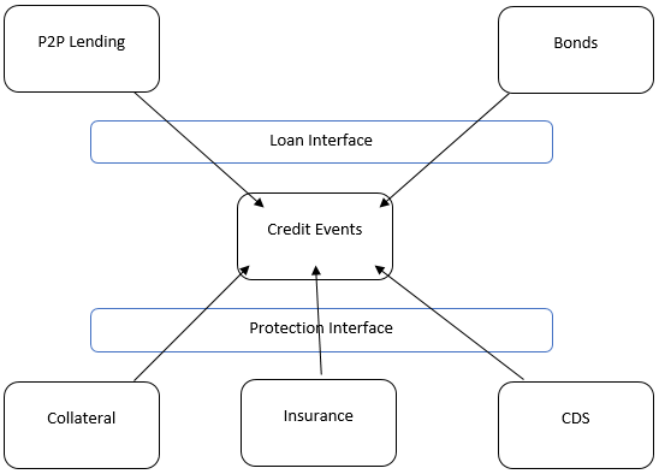


Fig. 2. The Ụgwọ lending infrastructure showing how the various loan and protection objects interface with the Credit Event object.

deployed and would not have to be worried about each loan that they insure individually. Protection would be external contracts and would just have a global view of all credit events from a single address given specific loan identifier (such as the loan address). To ensure compatibility, we can use interfaces which in object oriented programming specify the functions that must exist. Interfaces are similar to abstract classes in that they do not have any definition of functions contained within. An interface provides developers a guide as to how to implement the contract. Thus the Credit Event object is the core of extendable system where new loan types can be added and new protection types.

In Ụgwọ, we implement two interfaces: a Loan interface and Protection Interface. The Loan interface forces any loan object that would like to interact with the Credit Event object to implement certain functions that would enable the interaction. This same concept applies for the Protection Interface. These interfaces and their links to the loan objects are shown in Figure 2.

3.1 Overview of loan objects

Peer to peer lending. We start with a basic loan contract constructed by the cash provider. The loan has parameters such as the address of the cash provider and cash taker, the principal amount to be lent, the start and end dates of the loan, the repay value and repay schedule. The cash provider runs the constructor and funds the contract. The cash taker runs a function in the loan contract to retrieve the principal in the contract. At maturity, the borrower calls a function to pay back the principal with the corresponding interest. As with all of our

objects, modifiers ensure that only the stated party can run each a function in a contract, and an internal state machine governs at which phase of the contract each function can be called. If the borrower does not show up to retrieve the principal from the contract, the lender's money would remain the loan contract forever. To combat this, a kill function was implemented such that the lender can retrieve the money from the contract if the borrower does not retrieved the money after a timeout.

If the cash taker fails to pay back the loan within the timeframe, the loan object itself cannot transition states without someone calling a function. In Ụgwọ, a default function can be triggered by any person watching or monitoring the loan if the borrower fails to pay after the due term. This default function when run, updates the Credit Event object discussed below. This is how a loan moves into a default state and it relies one someone having an incentive to transition the loan (otherwise it is likely inconsequential if it sits dormant).

Bonds and commercial paper. We implement a simple 'zero coupon' bond. The contract uses an external library implementing EIP20 tokens.[4] The cash taker, generally an organization or corporation in this case, creates a set of tokens that represent units of cash it will accept (and later repay) from individual cash providers. The cash taker runs the constructor (with variables for start date, end date, bond value, repay value, *etc.*) and funds the contract with tokens. A function is used to accept payment from investors where tokens representing the amount borrowed is sent to the investors. The token is calculated as the value deposited over the price of the bond. An event is created that informs watchers of the contract of all bonds sold. The bond is a bearer bond in the sense that the bond contract does not track the addresses of who owns each bond. The token can be transferred from one person to another without interacting with the bond contract (however, the interaction is performed with the standard token contract). To get paid at maturity, only the token needs to be submitted irrespective of the bearer of the token. Defaults are implemented the same as in the P2P lending contract. The default function can be triggered by any person watching or monitoring the bond if the organization defaults on its payment after the due term.

3.2 Overview of protection objects

Collateral. Two types of collateral are defined in Ụgwọ—a token collateral and an ether collateral. A token collateral contract accepts a EIP20 token which might represent a token from a ICO, DAO-style contract, loan contract or anything else with value that the cash provider is willing to accept. The constructor function of the contract states the amount of tokens the cash taker is willing to put up as collateral. A separate function allows the cash taker to instantiate all agreements with the cash provider; they were not included in the constructor function to allow the collateral function to be run by any investor. If at the end

[4] https://github.com/ConsenSys/Tokens

of the term the cash taker defaults, a function to get the token out of escrow can be run by the cash provider. The function first checks for a credit event, or triggers a credit event if the conditions for a default are met. An ether collateral accepts ether as collateral—since the loan itself is in Ether, this is useful for partial collateral functions or when the collateral is backing insurance rather than a loan.

Credit default swaps. A credit default swap (CDS) is an agreement between two parties (a seller and a buyer) where the CDS seller fulfils the debt of a loan to the CDS buyer if a credit event occurs on the loan. The CDS seller then takes ownership of the loan. If more there is more than one seller of a CDS per loan (as is permitted and common in financial markets for speculation), the loan is auctioned and the market clearing price is used to settle the swaps.[5] A CDS seller subsumes the same risk position as the actual cash provider in the loan but the benefit to the CDS seller is not having to liquidate any assets (she can have effectively no cash on hand if an event never happens). The benefit to the cash provider is that a loan with a CDS only defaults if both the cash taker and the CDS seller default.

CDSs have a bad reputation after the 2008 financial crisis in the United States, where the CDS market was unlit and considered by many to be under-regulated. In Ụgwọ, the CDS market is transparent and CDS buyers can have enforced reserves that automatically settle with CDS buyers when a credit event occurs on an insured loan. CDS sellers themselves an be given a Credit Event object. Our implementation is rudimentary (without naked CDSes, auctions, or other features) and we expect that a full-fledged, decentralized CDS market would constitute an entire research paper by itself.

3.3 Overview of the CreditEvent object

It would be simpler to implement a CreditEvent object within each loan (P2P or Bond) contract. One reason to pull it out and make it an object of its own is to prevent redundancy in the use of code. This is a basic principle of object oriented programming. Another reason is to create a somewhat central place where all the loans can be monitored.

The simplest model of a CreditEvent object begins with a contract that holds all default variables such as defaulter's address, the lender's address and the defaulted amount. It implements a struct variable that is used to hold all the values pertaining to each loan. The contract implements the zero coupon payment model and hence has only one value for defaults. The value of the defaults could either be a string (yes or no) or a number (the amount defaulted). This contract has a constructor that is triggered by a loan contract. The major task of the constructor is to allocate memory for the loan that triggered it and set the necessary parameters (defaulter's address, the lender's address). An update function within the CreditEvent contract is triggered by loans to insert default

[5] http://www2.isda.org/

value into the struct variable. A defaultlist function acts as a getter function and returns all the values within the contract. This contract by itself performs no specific action beside receiving information from loans linked to it and acting as a global table visible to different protection objects and users.

In Ụgwọ, each loan's constructor triggers the CreditEvent function to insert arguments such as the lender's and debtor's address. A payback function contained within the loan is triggered by the debtor in other to pay back the principal and interest. It takes into factor the state of the contract as well as the maturity date of the loan. If the amount being paid by the debtor is less than the total amount (principal and interest), the amount is paid to the cash provider and a default written to the CreditEvent contract. A value of zero is written if the amount being paid covers the total amount or is in excess (in this case, the surplus is returned to the cash taker). A report function can be triggered by anyone watching the contract if the borrower defaults on its loan. This would set the loan to a default state such that anyone watching the loan can tell that the borrower defaulted on the loan.

4 Discussion

4.1 Exploring the use of oracles for exchange rates

It is not uncommon to encounter use cases that require a smart contract to trigger or change state in response to an event external to the blockchain. For example, an insurance contract might pay farmers based on the temperature and sunlight for a given period. A hypothetical smart contract might listen for any change in the weather, parse this information from an external source such as a URL, and then trigger payments or other events based on this information. As simple as this contract might sound, it is not possible to run contracts on Ethereum this way. This is because the blockchain follows a consensus-based model that ensures all inputs can be validated. Externally fetched data might differ between nodes, some nodes may not be able to access the data due to networking issues, and the amount of gas that should be consumed by the miner for spending time fetching the data is difficult to determine objectively.

In the case of our lending infrastructure, we want to implement a loan where the unit of account for the loan is based on the value of a fiat currency. The actual loan will be in Ether but the amount owed will be based on its current exchange rate with the underlying currency. This is side-step the monetary instability of Ether which makes it unattractive for lending. Thus in nominal terms, the amount of ether being paid back might be more or less than the amount borrowed depending on whether it's value increased or decreased relative to the fiat dollar. Bonds do not only offer an investment opportunity, but they allow investors to speculate or hedge on rates of inflation.

Since contracts cannot fetch external data, a service has emerged, called an oracle, which is trusted external entity that puts data onto the blockchain where it can be accessed by other contracts. In Ụgwọ, we use Oraclize[6] to feed the

[6] https://github.com/oraclize

exchange rate of Ether with USD into our contracts. Using an oracle is not foolproof and we note a few challenges in using an oracle. The first challenge is that the price is needed at each execution of the contract. Another challenge is that in order to feed the current exchange value into the blockchain, a link to any exchange has to be manually inserted into the oracle's code; if the link goes down, the oracle will not be able to provide the appropriate data into the blockchain to be used by the miners. Finally oracles are trusted parties that can lie about the exchange rate and collude with cash takers to steal from cash providers. We remark that oracles do have a reputation and in most countries, stealing is still subject to legal recourse even if it is on a blockchain.

4.2 Automatic actions

Many Ethereum beginners have to adjust their mental model of smart contracts to the fact that a contract will not run unless if one of its functions is called. It cannot automatically perform actions, say, after some period of time has passed. In Ụgwọ, loans like bonds have a default function that checks if there has been a default by the cash taker. This default function has to be triggered by someone in order to default the loan and update the CreditEvent object. An option is to use the Ethereum Alarm Clock [7] to trigger the function monthly. It is a trusted third party service that supports scheduling of transactions such that they can be executed at a later time on the Ethereum blockchain. This is done by providing all of the details for the transaction to be sent, an up-front payment for gas costs, which would allow your transaction to be executed on ones' behalf at a later time. The drawback is its heavy integration with the loan contract, as well as arranging payments to the service. Would it be possible for an actor in the loan contract to run the function monthly in other to avoid the heavy integration and cost of using the Ethereum alarm clock? Which actor in the loan contract would have a higher incentive to run the default function? All answers point towards the cash provider. Due to the fact that the insurance or collateral can only be claimed after a default occurs, the cash provider in the contract would have more incentive to run the function every month. Hence, we did not deploy the alarm clock.

4.3 Implementing the monthly array object

To implement a monthly payment, we could reference either time (*e.g.,* `now` or `block.timestamp`) or block interval (*e.g.,* `block.number`). Timestamps are not reliable and be manipulated by miners. This is due to the decentralization of the system; there is no wall clock for reference and node's local clocks can never be perfectly synchronized (*i.e.,* to the millisecond). Ethereum permits a 900 millisecond lead or lag in time. When using block numbers, there is also a lack of precision. One could estimate that a 31 day month would be something like

[7] `http://www.ethereum-alarm-clock.com/`

179 759 blocks.[8] While this is a challenge for applications that need near real-time fidelity but for loan payments, we would argue that time slippage is not critical for loans. We utilize time not blocks. If a loan lies dormant for longer than a month, with Ethereum's model of function-initiated state changes, the loan's state will not change. However the next function to be called, whether a payment or default check, will update the previously skipped months in CreditEvent while writing the current result of the called function.

4.4 Implementing the CreditEvent contract

Choosing an appropriate data structure for CreditEvent presented some challenges. We want loans to be individually encapsulated with the addresses of the cash provider and cash taker, and some data structure to hold a credit score for the loan (such as an array of values that indicate for each month whether the payment was repaid, late, defaulted, *etc.*). Note that it is not up to the CreditEvent object to penalize credit events. It passively records them and then protection objects can chose how to act. CreditEvent should be agnostic of what type of loan it is representing (*e.g.,* peer-to-peer, bond, *etc.*). In Ụgwọ, each bond is an individual loan. Protection objects, like credit default swaps, are generally written to monitor credit events across the entire issue of bonds, not just one individual bond. We leave for future work improvements to how sets of loans can be insured.

This credit history could be a struct, mapping or array. According to solidity documentation, in order to restrict the size of a struct, a struct is prevented from containing a member of its own type. However, the struct can itself be the value type of a mapping member. Following that, another way is to have a mapping to another struct outside of itself that contains the monthly defaults. In Solidity, mappings are like hash tables that are initialized dynamically with key/value pairs. Unmapped keys return an all zero byte-representation. However, it is not possible to iterate through the contents of a mapping and therefore, the best implementation was to have an array contained within a struct. In all cases, the inner container cannot be visible within the interface of the Ethereum wallet even if the outside container is made public. For example, if you implement a struct inside another struct and on, eventually the interface would give up trying to display all the subviews within it. To make the contract more developer friendly, we use getter functions to reach inside structs and expose the contents to the wallet interface.

In other to uniquely identify loans in the CreditEvent contract, when a loan calls the CreditEvent contract to pass in the initial parameters, a loan id number is created by the CreditEvent contract. This loan id number can be used by a protection object to monitor a loan. Using a loan id number creates an extra variable that floats around the contract that might not necessarily be needed. A better approach is to use the loan address as a unique identifier. This way the protection object do not need to keep the loan id number of every loan

[8] Blocks 4652926 to 4832685 were mined in December 2017.

they monitor as the address of the loan by itself serves as a unique identifier. This however, is not a hard rule as either a loan ID number or address can be used to uniquely identify a loan without causing any mishap in general. Even in situations where two loans are created at the same time, the id of the loans is set by the miner in the order in which they are place within the block. The interface of the Ethereum wallet for the CreditEvent object contains the parameters for identifying each loan on the CreditEvent object. The loan address is used to retrieve this information. The months which have no default are represented with zero and 300000000000000000 wei (0.3 ether) is the default amount for the second month. To pay out this default, any protection object would just need to fetch the value from the CreditEvent object.

4.5 Implementing a Credit Default Swap

In the implementation of loan insurance, such as through a credit default swap, it is ironic that such a contract operates as both a protection object against counter-party risk and also introduces new counter-party risk (that the insurer will default on paying the insurance). The CDS contract is drawn up by the CDS seller who initializes agreed upon facts such as the CDS buyer, amount to be insured, premium, among others. During the payment by the CDS buyer, the function allocates space in the CreditEvent object to hold information regarding the standings of payments made to the CDS buyer.

If a default occurs on a loan that has been insured with a CDS, the default function would be run by the CDS buyer (the buyer has a higher stake and more incentive to run the function). This function would update the CreditEvent object with the balance of the loan to be paid. This is because when a default occurs, the rest of the debt is paid to the CDS buyer and the CDS seller takes over the loan (this is where the swap occurs). The idea behind this is that we wanted the CDS contract to fetch the balance of the debt directly from the CreditEvent object just as the Collateral object gets the default for the month from the CreditEvent object and pays out to the cash provider. This way the amount to be paid cannot be manipulated by either the CDS seller or anyone and the payment can be made automatically when triggered.

When the payment is made to the CDS buyer, a change of ownership occurs. This could be implemented in two ways. One way is to have a new contract created for the change of ownership where the CDS seller becomes the Lender in the loan contract. This would create a new contract which might be hard to track as it would have a new address with no relation to the old address. The other way, which we implemented, is to have the same loan contract implemented for the CDS change the owner name. This way the new owner (CDS seller) is tied to the loan contract and anyone who had the address for watching the CDS loan would be aware that a credit swap occurred. The change of ownership is also reflected in the CreditEvent object.

Contract	Gas	Ether	USD
Base System			
Tokens	857,106	0.018	$5.00
Token Transfer	51,501	0.001	$0.30
Oraclized	154,711	0.003	$0.90
Credit Score	462,453	0.010	$2.70
Peer to Peer Lending			
P2P Lending	2,198,423	0.046	$12.82
Receive Money	474,112	0.009	$2.77
Payback	105,827	0.002	$0.62
Report Default	60,605	0.001	$0.35
Kill	25,098	0.001	$0.12
Bond			
Bond	2,229,084	0.047	$13.00
Purchase Bond	231,397	0.005	$1.35
Withdraw	292,787	0.006	$1.71
Repay	415,213	0.009	$2.42
Report Default	55,798	0.001	$0.33

Table 1. Cost of running the basic and loan contracts

5 Evaluation

Our contracts were developed in Remix and tested on Ethereum's test network.

5.1 Security

Solidity (and Serpent) is notorious for security issues [6,8,2]. We made our contract resilient to the re-entrancy bug by ensuring that all checks are performed before transfers (such as, does the sender have enough ether?) and also ensuring that state variables are changed before transfers. Mishandled exceptions have the potential to allow unauthorized access to functions or result in denial of service attacks on individual smart contracts. We handle this in our contracts with the use of modifier functions that act as an access control mechanism. This allows only authorized users to access functions and also sanitizes inputs to reduce the likelihood of exceptions. Transaction-ordering dependence and timestamp dependence attacks do not break our contract due to the nature of our project. Although timestamps (as opposed to block numbers) are used in our project, our contract is not time dependent and any modification of the time by factor of 900 seconds by the miner will not break the contract. Last, the price for a bond in our system is fixed by the bond issuer and cannot be changed after deployment. Therefore, the contracts are not susceptible to a transaction ordering attacks.

Contract	Gas	Ether	USD
Collateral			
Collateral	442,035	0.009	$2.58
Serve	204,509	0.004	$1.20
Get Ownership	312,667	0.007	$1.82
Cancel	27,664	0.001	$0.16
Credit Default Swap			
CDS Contract	452,035	0.009	$2.58
Monthly Premium	204,509	0.004	$1.20
Report Default	61,709	0.001	$0.16
Kill	27,664	0.001	$0.16

Table 2. Cost of running the protection contracts

In order to test our system for known security bugs, we use a symbolic execution tool called Oyente [8].[9] The tool has been proved in successfully identifying critical security vulnerability, such as a famous incident called the DAO vulnerability. The various APIs used by both contracts were analyzed together simulating the exact same way it would be deployed. None are vulnerable to any of the tests.

5.2 Cost

In this section we would analyze the gas cost of using our contracts. As of this writing, the current price per gas is 21 gwei (0.000000021 Ether) while the current price of 1 ether = $277.78. For any contract, the gas cost = gas * gas price. As of this writing, it is useful to note that any transfer of ether from one account to another has a gas of 21,000, a gas cost of 0.00044 Ether resulting to $0.12 USD. Table 1 and 2 represents the cost of running each smart contract and its functions contained therein on the Ethereum Virtual Machine. The cost of deploying the P2P lending contract and the Bond contract is roughly about $13.00 respectively. This is due to the API's called by those contracts, the more API's a contract import the more the code needed to be executed by the miners and the higher the gas consumption. In particular, the high gas consumption is attributed to the Oraclized API. However, once deployed, the cost of running the rest of the function inside the contract is less than $3.00.

5.3 Concluding Remarks

We have present Ụgwọ, an Ethereum implementation of a lending infrastructure. We use the term infrastructure because Ụgwọ is not a single system, but rather a central component (CreditEvent) with two interfaces for an extensible system,

[9] https://github.com/ethereum/oyente

where new loan and loan protection techniques can be added. Future work might deploy more exotic bonds or commercial paper arrangements, or other types of protection techniques like reputation systems and repurchase agreements.

Acknowledgements. J. Clark acknowledges funding for this work from NSERC and FQRNT.

References

1. M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure Multiparty Computations on Bitcoin. In *IEEE Symposium on Security and Privacy*, 2014.
2. N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (sok). In *POST*, 2017.
3. I. Bentov and R. Kumaresan. How to Use Bitcoin to Design Fair Protocols. In *CRYPTO*, 2014.
4. G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *NDSS*, 2015.
5. C. Decker and R. Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *SSS*, 2015.
6. K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *BITCOIN*, 2016.
7. T. Latter. The choice of exchange rate regime. In *Centre for Central Banking Studies*, volume 2. Bank of England, 1996.
8. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *CCS*, 2016.
9. J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Technical Report (draft). https://lightning. network, 2015.
10. T. Ruffing, A. Kate, and D. Schröder. Liar, liar, coins on fire!: Penalizing equivocation by loss of bitcoins. In *CCS*, 2015.
11. F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. Town crier: An authenticated data feed for smart contracts. In *CCS*, 2016.