Formal Verification of Smart Contracts

Chad E. Brown[a] Ondřej Kunčar[b] Josef Urban[a]

[a] *Czech Technical University in Prague (CIIRC), Prague, Czech Republic*

[b] *Fakultät für Informatik, Technische Universität München, Munich, Germany*

In the recent years, cryptocurrencies have seen steady increase of popularity and adoption as witnessed by the most popular cryptocurrency Bitcoin. Bitcoin's main appeal—no need for a trusted central authority—is achieved by a shared data structure (blockchain) and a distributed consensus protocol. These two technologies can be further exploited to go beyond a traditional notion of a currency (i.e., sending money from A to B) by running programs on a blockchain such that their correct execution is enforced again without any centralized authority. These programs are called *smart contracts*.

Ethereum [11] is a flagship of cryptocurrencies that allow for smart contracts written in a Turing-complete programming language. Many smart contracts, such as financial derivatives, prediction markets or auctions, have been implemented in Ethereum since its release. But with popularity comes pressure: in June 2016, an attacker managed to transfer approx. $60M from TheDAO contract into their control due to a programming error in the contract. In our project, we propose a holistic solution to security of smart contracts by combining two technologies: distributed proof market and program verification in an interactive theorem prover (ITP).

Distributed Proof Market

Writing proofs (regardless if for mathematical statements or program verification) is a difficult task and quite often one has to invest considerable energy in finding and organizing manpower. The seminal Flyspeck project (the formal proof of the Kepler conjecture in HOL Light and Isabelle/HOL) required to centrally administrate many proof engineers to write 500000 lines of proofs over a span of 20 person-years.

To ease this problem, we propose to implement a proof market, a tool that allows for i) setting up a bounty for a theorem statement; ii) rewarding the first author that submits a valid proof. Such a tool gets useful only then when the system is decentralized, i.e., the rewards could be collected without any interaction from the problem creator.

This can be achieved by using smart contracts. The contract would be a simple proof checker that would reward anybody that submits a valid proof term for the published problem. By combining this with a certain form of a commitment scheme, we would guarantee

that the *first* author gets rewarded, i.e., the proof cannot be stolen between it gets submitted and the network reaches consensus (i.e., the reward is paid).

We have started to explore Ethereum implementation of a very simple proof checker, such as the Metamath [9] verifier. [1] It consists of simple stack and string operations implemented in 300 lines of Python. Our initial estimate is that verifying the whole Metamath set-base development of 20000 toplevel lemmas would take about 100 million string operations. An implementation of this in Ethereum (or a variant) seems to be feasible. The current cost of 100 million string operations differs across different Ethereum versions and over time. In Ethereum today it is $2M, and in 2015's Ethereum this would be about $20000. These prices are high but there is active research in collaborative execution of smart contracts which could significantly lower these prices.

Earlier related work on "outsourcing proofs" include (i) ProofMarket [5], which is centralized, (ii) the first author's Mathgate [2], which is decentralized but the reward was not given for proving a theorem but for giving one particular proof of a theorem, and (iii) Qeditas [10], which seems to be work-in-progress, unlike Ethereum.

Program Verification in an ITP

Ethereum smart contracts are usually implemented in a high-level language (Solidity is the most prominent one) and get compiled into Ethereum virtual machine bytecode (EVM). A couple of approaches have been suggested to obtain some correctness guarantees [1,3,8] but they rather offer a verification of special properties than full flexibility of a theorem prover.

We suggest the following methodology: use the specification language of the interactive theorem prover Isabelle/HOL as the high-level language and specify the contract abstractly (using mathematical concepts such as functions or sets) and conduct the high-level proofs on this level (e.g., "Can clients lose money?"). Then refine the program to a more low-level implementation in a stepwise fashion and tackle the low-level properties (e.g., "Can the stack overflow?")—each step is a theorem relating two programs. At the end one obtains list of bytes (deep embedding of the EVM bytecode in Isabelle [6]) and a theorem relating the abstract version to the corresponding EVM bytecode and thus its correctness. Many parts of this process can be automated by Isabelle's Refinement Framework [7], which has been proved to be useful in many projects such a verification of an LTL model checker [4].

We are currently exploring proving the correctness of some simple smart contracts from literature [3, 8].

Combining Both Technologies

When our methodology gets interesting is when we combine both technologies. One can set up a bounty to prove their smart contract correct, somebody else carries out the proof in Isabelle and exports the corresponding proof term and eventually gets rewarded without any interaction with the smart contract author. We believe that the achieved synergy between these two technologies can foster their further development.

In our poster, we would like to present our whole scheme of a distributed proof market and a program verification in Isabelle in more detail and report on our preliminary results with simple proof checkers and verification of simple smart contracts.

[1] http://us.metamath.org/downloads/mmverify.py

References

[1] Bhargavan, K., A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy and S. Zanella-Béguelin, *Formal verification of smart contracts: Short paper*, in: *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16, 2016, pp. 91–96.

[2] Brown, C. E., *The Egal manual* (2014).
URL http://mathgate.info/egalmanual.pdf

[3] Delmolino, K., M. Arnett, A. E. Kosba, A. Miller and E. Shi, *Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab*, IACR Cryptology ePrint Archive **2015** (2015), p. 460.
URL http://eprint.iacr.org/2015/460

[4] Esparza, J., P. Lammich, R. Neumann, T. Nipkow, A. Schimpf and J. Smaus, *A fully verified executable LTL model checker* (2013), pp. 463–478.

[5] Hirai, Y., *proofmarket*.
URL http://proofmarket.org

[6] Hirai, Y., *Defining the ethereum virtual machine for interactive theorem provers*, in: *1st Workshop on Trusted Smart Contracts*, 2017.

[7] Lammich, P., *Automatic data refinement*, in: *Interactive Theorem Proving - 4th International Conference, ITP 2013*, 2013, pp. 84–99.

[8] Luu, L., D. Chu, H. Olickel, P. Saxena and A. Hobor, *Making smart contracts smarter*, in: *Proceedings of the Conference on Computer and Communications Security 2016*, 2016, pp. 254–269.

[9] Megill, N., "Metamath: A Computer Language for Pure Mathematics," Lulu Press, Morrisville, North Carolina.

[10] White, B., *Qeditas: A formal library as a bitcoin spin-off* (2016).
URL http://qeditas.org/docs/qeditas.pdf

[11] Wood, G., *Ethereum: A secure decentralised generalised transaction ledger* (2004).
URL http://yellowpaper.io