VeriSolid for TRANSAX: Correct-by-Design Ethereum Smart Contracts for Energy Trading

Aron Laszka
University of Houston

Anastasia Mavridou
KBR / NASA Ames Research Center

Scott Eisele
Vanderbilt University

Emmanouela Stachtiari
Aristotle University of Thessaloniki

Abhishek Dubey
Vanderbilt University

1 INTRODUCTION

The adoption of blockchain based platforms is rising rapidly. Their popularity is explained by their ability to maintain a distributed public ledger, providing reliability, integrity, and auditability without a trusted entity. Recent platforms, e.g., Ethereum, also act as distributed computing platforms and enable the creation of smart contracts, i.e., software code that runs on the platform and automatically executes and enforces the terms of a contract. Since smart contracts can perform any computation, they allow the development of decentralized applications, whose execution is safeguarded by the security properties of the underlying platform. Due to their unique advantages, blockchain based platforms are envisioned to have a wide range of applications, ranging from financial to the Internet-of-Things.

However, the trustworthiness of the platform guarantees only that a smart contract is executed correctly, not that the code of the contract is correct. In fact, a large number of contracts deployed in practice suffer from software vulnerabilities, which are often introduced due to the semantic gap between the assumptions that contract writers make about the underlying execution semantics and the actual semantics of smart contracts. A recent automated analysis of 19,336 smart contracts deployed in practice found that 8,333 of them suffered from at least one security issue. Although this study was based on smart contracts deployed on the public Ethereum blockchain, the analyzed security issues were largely platform agnostic. Security vulnerabilities in smart contracts present a serious issue for two main reasons. Firstly, smart-contract bugs cannot be patched. By design, once a contract is deployed, its functionality cannot be altered even by its creator. Secondly, once a faulty or malicious transaction is recorded, it cannot be removed from the blockchain ("code is law" principle). The only way to roll back a transaction is by performing a hard fork of the blockchain, which requires consensus among the stakeholders and undermines the trustworthiness of the platform.

In light of this, it is crucial to ensure that a smart contract is secure before deploying it and trusting it with significant amounts of cryptocurrency. To this end, we present the VeriSolid framework for the formal verification and generation of contracts that are specified using a transition-system based model with rigorous operational semantics [7]. VeriSolid provides an end-to-end design framework, which combined with a Solidity code generator, allows the correct-by-design development of Ethereum smart contracts. To the best of our knowledge, VeriSolid is the first framework to promote a model-based, correctness-by-design approach for blockchain-based smart contracts. Properties established at any step of the VeriSolid design flow are preserved in the resulting smart contracts, guaranteeing their correctness. VeriSolid fully automates the process of verification and code generation, while enhancing usability by providing easy-to-use graphical editors for the specification of transition systems and natural-like language templates for the specification of formal properties. By performing verification early at design time, VeriSolid provides a cost-effective approach since fixing bugs later in the development process can be very expensive. Our verification approach can detect typical vulnerabilities, but it may also detect any violation of required properties. Since our tool applies verification at a high-level, it can provide meaningful feedback to the developer when a property is not satisfied, which would be much harder to do at bytecode level.

We present the application of VeriSolid on smart contracts used in Smart Energy Systems such as transactive energy platforms. In particular, we used VeriSolid to design and generate the smart contract that serves as the core of the TRANSAX blockchain-based platform for trading energy futures [3]. The designed smart contract allows energy producers and consumers to post offers for selling and buying energy. Since optimally matching selling offers with buying offers can be very expensive computationally, the contract relies on external solvers to compute and submit solutions to the matching problem, which are then checked by the contract. Using VeriSolid, we defined a set of safety properties and we were able to detect bugs after performing analysis with the NuSMV model checker.

2 VERISOLID FRAMEWORK

VeriSolid [5, 6] is an open-source[1] and web-based framework that is built on top of WebGME [4]. VeriSolid allows the collaborative development of Ethereum contracts with built-in version control, which enables branching, merging, and history viewing. Figure 1 shows the steps of the VeriSolid design flow. Mandatory steps are represented by solid arrows, while optional steps are represented by dashed arrows. In step (1), the developer input is given, which consists of:

- A contract specification containing 1) a graphically specified transition system and 2) variable declarations, actions, and guards specified in Solidity.
- A list of properties to be verified, which can be expressed using predefined natural-language like templates.

BDLT'19, September 2–5, 2019, Vienna, Austria
2019.

[1] https://github.com/anmavrid/smart-contracts

Table 1: Analyzed properties and verification results for TRANSAX

Properties	Type	Result
(i) if `close` happens, `postSellingOffer` or `postBuyingOffer` can happen only after `finalize.offers.length=0`	Safety	Violated
(ii) `register.prosumers[msg.sender]= prosumerID` cannot happen after `setup`	Safety	Verified
(iii) `register` cannot happen after `setup`	Safety	Verified
(iv) if `finalize` happens `createSolution` or `addTrade` can happen only after `close`	Safety	Verified

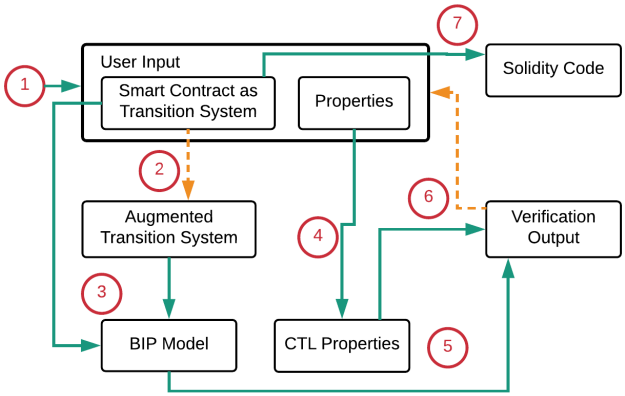


Figure 1: Design and verification workflow.

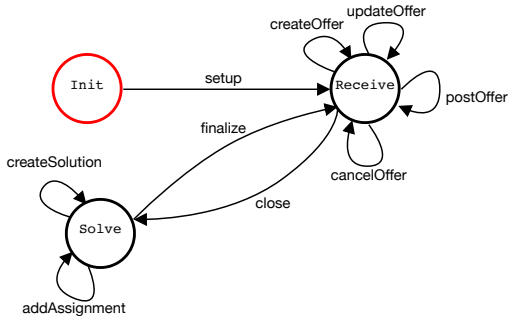


Figure 2: VeriSolid model of the TRANSAX smart contract.

The verification loop starts at the next step. Optionally, step (2) is automatically executed if the verification of the specified properties requires the generation of an augmented contract model. Next, in step (3), the model of the contract in the Behavior-Interaction-Priority (BIP) formal language [1] is automatically generated. Similarly, in step (4), the specified properties are automatically translated to Computational Tree Logic (CTL). The model can then be verified for deadlock freedom or other properties using tools from the BIP toolchain [1] or nuXmv [2] (step (5)). If the required properties are not satisfied by the model (depending on the output of the verification tools), the specification can be refined by the developer (step (6)) and analyzed anew. Finally, when the developers are satisfied with the design, i.e., all specified properties are satisfied, the equivalent Solidity code of the contract is automatically generated in step (7).

3 TRANSAX PLATFORM

Power grids are undergoing major changes due to rapid growth in renewable energy and improvements in battery technology. Prompted by the increasing complexity of power systems, decentralized transactive solutions are emerging, which arrange local communities into transactive microgrids. The core functionality of transactive microgrids is to provide an efficient market that matches producers of energy with consumers, while ensuring the safety of the power system and the privacy of the participants.

TRANSAX is a smart contract based energy-trading platform for transactive microgrids [3]. TRANSAX provides safety by ensuring the power line capacity constraints are respected by trading, and it provides privacy through an anonymizing mixer. To efficiently match producers and consumers, TRANSAX uses a hybrid architecture, which combines the trustworthiness of smart contracts with the efficiency of conventional computational platforms.

4 VERISOLID FOR TRANSAX

We have generated the correct-by-design Solidity code of the TRANSAX smart contract using VeriSolid. The initial (before the augmentation) VeriSolid transition system of the contract is shown in Figure 2. The contract has three states:

- `Init`, in which the contract has been deployed but not been initialized. Before the contract can be used, it must be initialized (i.e., numerical parameters must be set up).
- `Receive`, which corresponds to the *offering* phase of a cycle. In this state, prosumers may post (or cancel) their offers.
- `Solve`, which corresponds to the *solving* phase of a cycle. In this state, solvers may submit solutions (i.e., resource allocations) based on the posted (but not cancelled) offers.

We defined a set of safety properties for this contract (Table 1 presents a subset of these properties). We were able to find a bug in the action of the `finalize` transition:

```
// action of finalize transition
if (solutions.length > 0) {
    Solution storage solution = solutions[bestSolution];
    for (uint64 i = 0; i < solution.numTrades; i++) {
        Trade memory trade = solution.trades[i];
        emit TradeFinalized(trade.sellingOfferID,
        trade.buyingOfferID, trade.power, trade.price);
    }
    solutions.length = 0;
    offers.length = 0;
}
// offers.length = 0; SHOULD HAVE BEEN HERE
cycle += 1;
```

This bug was immediately detected as a violation of our first safety property shown in Table 1.

REFERENCES

[1] Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.-H., and Sifakis, J. Rigorous component-based system design using the bip framework. *IEEE Software 28*, 3 (2011), 41–48.

[2] Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., and Wang, Q. Formal verification of infinite-state BIP models. In *Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA)* (2015), Springer, pp. 326–343.

[3] Laszka, A., Eisele, S., Dubey, A., and Karsai, G. TRANSAX: A blockchain-based decentralized forward-trading energy exchange for transactive microgrids. In *Proceedings of the 24th IEEE International Conference on Parallel and Distributed Systems (ICPADS)* (December 2018).

[4] Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., and Lédeczi, Á. Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure. In *Proceedings of the MPM@ MoDELS* (2014), pp. 41–60.

[5] Mavridou, A., and Laszka, A. Designing secure Ethereum smart contracts: A finite state machine based approach. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)* (February 2018).

[6] Mavridou, A., and Laszka, A. Tool demonstration: FSolidM for designing secure Ethereum smart contracts. In *Proceedings of the 7th International Conference on Principles of Security and Trust (POST)* (April 2018).

[7] Mavridou, A., Laszka, A., Stachtiari, E., and Dubey, A. VeriSolid: Correct-by-design smart contracts for Ethereum. In *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC)* (February 2019).

VeriSolid for TRANSAX: Correct-by-Design Ethereum Smart Contracts for Energy Trading

Aron Laszka, Anastasia Mavridou, Scott Eisele, Emmanouela Stachtiari, Abhishek Dubey

VANDERBILT UNIVERSITY
INSTITUTE FOR SOFTWARE INTEGRATED SYSTEMS

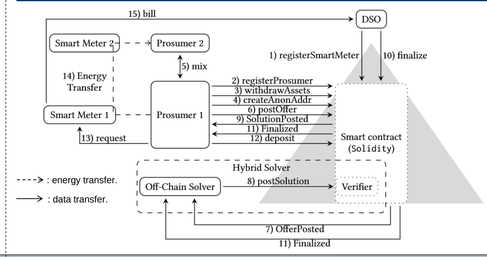
Background

Distributed ledgers are immutable by design
- smart contract vulnerabilities **cannot be patched**
- erroneous (or malicious) transactions **cannot be reverted**

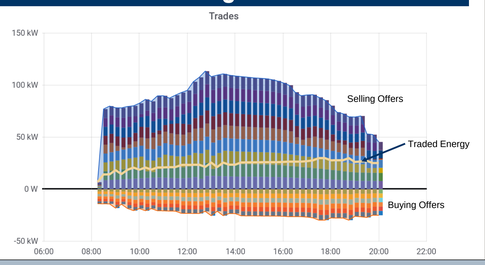
Recent analysis: 34,200 contracts (out of 1M publicly deployed contracts) have security issues / vulnerabilities

Three main approaches for securing smart contracts:
- Design patterns
- Tools for finding (typical) vulnerabilities
- Tools for verification and static analysis

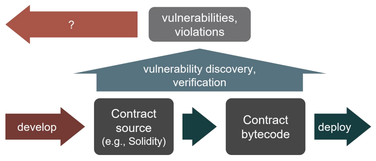
TRANSAX : Safe and Private Forward-Trading Platform for Transactive Microgrids



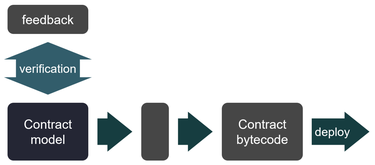
Energy trading markets are safety-critical systems and cannot afford errors in smart contract-based logic.



Correct-by-Design Contract Development

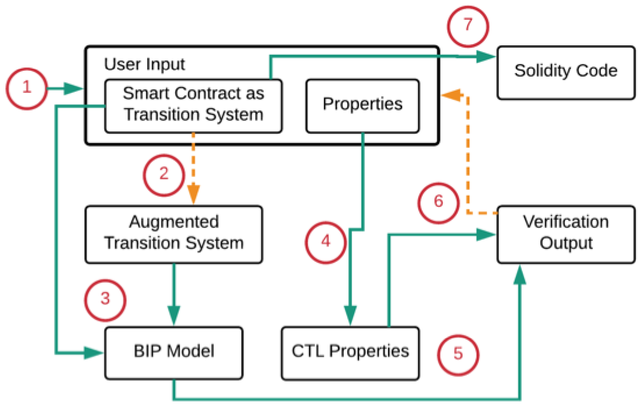


Traditional approach



VeriSolid approach

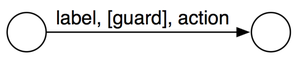
Advantages of model-based approach:
- Specification of **desired properties** with respect to a **high-level model**
- Providing **feedback** to developer with respect to a **high-level model**



VeriSolid Language

Model

- Formal, transition-system based language for contracts



- Each contract may be represented as a transition system
- A smart contract is a tuple $(D, S, S_F, s_0, a_0, a_F, V, T)$
  - D custom data types and events
  - S states
  - $S_F \subset S$ final states
  - $a_0 \subset \mathbb{S}$ initial action
  - $a_F \subset \mathbb{S}$ fallback action
  - V contract variables
  - T transitions (names, source and destination states, actions, guards, parameter and return types)

$\mathbb{S}$: subset of Solidity statements

Implemented as functions in the generated code

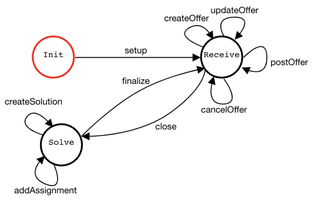
Formal Semantics

- Example transition rule:

$$\text{TRANSITION}\quad \frac{\begin{array}{c} t \in T, \quad name = t^{name}, \quad s = t^{from} \\ M = Params(t, v_1, v_2, \ldots), \quad \sigma = (\Psi, M) \\ \text{Eval}(\sigma, g_t) \rightarrow \langle(\hat{\sigma}, N), \texttt{true}\rangle \\ \langle(\hat{\sigma}, N), a_t\rangle \rightarrow \langle(\hat{\sigma}', N), \cdot\rangle \\ \hat{\sigma}' = (\Psi', M'), \quad s' = t^{to} \end{array}}{\langle(\Psi, s), name\,(v_1, v_2, \ldots)\rangle \rightarrow \langle(\Psi', s', \cdot)\rangle}$$

Verification of the TRANSAX Contract

- Instead of searching for vulnerabilities, we verify that a model satisfies **desired properties** that capture **correct behavior**
- **Deadlock freedom**: contract cannot enter a non-final state in which there are no enabled transitions
- **Safety** and **liveness properties**
  - Specified using Computational Tree Logic (CTL)
  - We provide several CTL templates to facilitate specification



TRANSAX: smart contract-based energy-trading platform for transitive microgrids

Violated Property Example

If `close` happens, `postSellingOffer` or `postBuyingOffer` can happen only after `finalize.offers.length=0`

```
// action of finalize transition
if (solutions.length > 0) {
    Solution storage solution = solutions[bestSolution];
    for (uint64 i = 0; i < solution.numTrades; i++) {
        Trade memory trade = solution.trades[i];
        emit TradeFinalized(trade.sellingOfferID,
        trade.buyingOfferID, trade.power, trade.price);
    }
    solutions.length = 0;
    offers.length = 0;
}
// offers.length = 0; SHOULD HAVE BEEN HERE
cycle += 1;
```

Conclusions

VeriSolid advantages
- High-level model with **formal semantics** (which are familiar to most developers)
- Verification of **desired behavior** (instead of searching for typical vulnerabilities)
- High-level **feedback** to the developer (for violated properties)
- Solidity **code generation** (instead of error-prone coding)

Source code: http://github.com/anmavrid/smart-contracts

Live demo at: http://cps-vo.org/group/SmartContracts
(requires free registration)