Smart contracts in a bare-bone UTXO model

Massimo Bartoletti[1], Riccardo Marchesin[2], Roberto Zunino[2]
[1]Università degli Studi di Cagliari, [2]Università degli Studi di Trento

Abstract

We present a framework for smart contracts in the UTXO-based model. Our approach allows for expressive smart contracts written in a high-level language, which are securely compiled into bare-bone UTXO transactions.

Most mainstream blockchains today follow the account-based model: e.g., besides Ethereum, also Avalanche, Hedera, and Algorand are account-based. For most developers, it is natural to interpret contracts as objects with a state that can be accessed and modified by methods. Account-based blockchains directly enable this programming style, which explains their dominance as smart contract platforms. Instead, in the UTXO-based model, the global state of the blockchain is represented by the set of unspent transaction outputs. Each output stores crypto assets and a state, and it specifies a redeem script which sets the conditions under which the output can be spent. A transaction can spend one or more outputs, specifying them as its inputs: this effectively removes these outputs from the global state, and creates new ones. These new outputs can update the state of the spent ones and redistribute the assets according to the redeem script. Despite programming UTXO-based contracts requires a change of mind from the common object-oriented style, the UTXO model has a series of advantages over the account-based model.

A first problem of Ethereum-style contracts is that their execution is not easily parallelizable over multi-core validators. Two transactions could be executed concurrently when their execution does not access the same part of the state. In Ethereum, validators have no efficient way to detect when two transactions are concurrent, because, in general, determining the part of the state accessed by transactions requires to execute it. In the UTXO-based model instead it is straightforward to detect when two transactions are concurrent: this just requires to check whether they spend disjoint outputs.

Another problem of the account-based model is that a user sending a transaction to the mempool can not precisely predict the state in which it will be executed. This has several negative consequences, such as the unpredictability of transaction fees and susceptibility to Maximal Extractable Value (MEV) attacks. In Ethereum-style contracts, a valid transaction must carry a gas fee which depends on the instructions needed to execute it. The actual number, type and cost of these instructions heavily depends on the initial state. Besides

forcing users to over-approximate the amount of gas needed, this also enables attacks in which an adversary front-runs a user transaction, so that it is executed in a state where the provided gas is insufficient. This makes the user pay the gas fee even if the transaction is rejected and does not update the state according to the user intentions. With MEV attacks instead, malicious validators manage to have an improper gain by reordering the users' transactions in the mempool and interleaving them with their own. Such attacks are very common in practice targeting in particular DeFi contracts, for an estimated value exceeding USD 700 million [1].

Unlike in the account-based model, in the UTXO-based model when a user sends a transaction T to the mempool, they know *exactly* the state in which it will be executed: indeed, this state is completely determined by T's inputs. Therefore, if an adversary front-runs T with another transactions T', the effect is that T will become invalid, preventing it to be appended to the blockchain, since some of its inputs are spent by T'. This avoids both issues described before, i.e. MEV attacks and unpredictability of fees. If the user still desires to perform the action in the new state, they must resend T, updating its inputs (and therefore, specifying the new state where the action is executed).

Currently, the two main blockchain platforms following the UTXO-based model are Bitcoin and Cardano. These two platforms have substantially different scripting languages, which results in smart contracts with different expressiveness. On the one hand, Bitcoin has a minimalistic scripting language, featuring only basic arithmetic and logical operations, conditionals, hashes, and (limited) signature verification [4]. While some interesting classes of smart contracts are expressible in Bitcoin [3], contracts requiring unbounded computational steps, or transfers of tokens different than the BTC, cannot be expressed. Neglecting the lack of expressiveness, this design choice has some positive aspects: besides limiting the attack surface and simplifying the overall design (e.g., no gas mechanism is needed), it makes contracts amenable to formal verification. On the other side of the spectrum, Cardano's scripting language is an untyped lambda-calculus [8], which makes its contracts Turing-complete. This increase in expressiveness comes at a cost, in that the static verification of general script properties is undecidable. A relevant research question is then whether one can find a balance between the two approaches, i.e. a contract model which is expressive enough for real-world use cases, without requiring a Turing-complete script language.

In this talk we address this question, by proposing an UTXO-based contract model which allows for expressive (Turing-complete) contracts, by only requiring a minimal (*non* Turing-complete) scripting language. The key idea is to scatter the execution of complex contract actions across multiple transactions. Even though each of these transactions only executes a simple (loop-free) script, the overall chain of transactions can encompass complex (possibly, recursive) behaviours. In this way, our model can support real-world use cases, like e.g. auctions and Automated Market Makers.

The talk will touch the following points:

- We sketch a stack of three languages. The top layer is a high-level contract language inspired by Solidity, the main language of Ethereum. The intermediate layer is a process calculus with basic primitives to manage crypto-assets. The bottom layer is an UTXO-based model with Bitcoin-like scripts extended with covenants [5–7].
- We discuss how to compile the high-level language into our intermediate language.
- We overview a *secure compiler* from the intermediate language to our UTXO language. Technically, the security of the compiler is established through a *computational soundness* theorem [2]: namely, even in the presence of adversaries, with overwhelming probability there is a step-by-step correspondence between the execution of an intermediate-level contract and that of the low-level contract resulting from its compilation.

References

[1] MEV-explore: MEV over time, March 2023. explore.flashbots.net.

[2] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.

[3] Nicola Atzei, Massimo Bartoletti, Stefano Lande, Nobuko Yoshida, and Roberto Zunino. Developing secure Bitcoin contracts with BitML. In *ESEC/FSE*, 2019.

[4] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. A formal model of Bitcoin transactions. In *Financial Cryptography and Data Security*, volume 10957 of *LNCS*, pages 541–560. Springer, 2018.

[5] Massimo Bartoletti, Stefano Lande, and Roberto Zunino. Bitcoin covenants unchained. In *ISoLA*, volume 12478 of *LNCS*, pages 25–42. Springer, 2020.

[6] Malte Möser, Ittay Eyal, and Emin Gün Sirer. Bitcoin covenants. In *Financial Cryptography Workshops*, volume 9604 of *LNCS*, pages 126–141. Springer, 2016.

[7] Russell O'Connor and Marta Piekarska. Enhancing Bitcoin transactions with covenants. In *Financial Cryptography Workshops*, volume 10323 of *LNCS*. Springer, 2017.

[8] Plutus Team. Formal specification of the plutuscore language, 2022.