SCRIFY : an online tool to validate Bitcoin scripts

Stefano Bistarelli[1], Andrea Bracciali[2], and Ivan Mercanti[1]

[1]University of Perugia
[2]University of Stirling

The Bitcoin blockchain records chained transactions of tokens from *source* to *destination* accounts. Such accounts are not explicitly linked to the owners' identity but to a cryptography key, which can be freely - and secretly - generated by any party. Each transaction has associated an *input script* and *an output script*. When a new transaction is proposed, the input script provides the needed credentials to unlock funds. The output script of the previous transaction that sourced the account validates the provided credentials according to a chosen scheme.

Scripts are written in a precise way; a stack-based scripting language called SCRIPT. Striving for correctness, robustness and efficiency in such an unconventional and constrained execution model, SCRIPT has been designed according to minimality principles, e.g. it is not Turing complete, has no recursion, cycles or procedure calls, has an execution cost (to be paid by the transaction proposer) proportional to the length of the code, and "dangerous" operations, like multiplication, are not allowed.

We provide a solution to the problem of the *satisfiability of "open" output scripts*, i.e. given an output script, which information has to be provided by an input script to let the output script run successfully and validate the transition? Although the simulation and execution of *"closed"* input-output scripts present, no problems and many tools and simulators are available[1], we observe that verification frameworks for the satisfiability of available scripts that cover a substantial fragment of the language, are up-to-date with the latest SCRIPT improvements and may modularly scale up to decentralised applications, are not so widespread.

The execution of SCRIPT code of an output script is simulated from an empty stack according to symbolic semantics. The initial stack for a successful computation is defined via a lazy approach collecting the weakest constraints on initial data over successful computations (further details in [2]). On top of that, we implemented a SCRIPT symbolic verification in SCRIFY (SCRIpt veriFY), an

[1]`https://siminchen.github.io/bitcoinIDE/build/editor.html`.

open source application implemented in Haskell, with a Prolog constraint solver and distributed as a Docker component [1].

Given an *output script* S, the current version of the tool returns all the existing satisfiable Γs for each successful computation of S. Such Γs are specifications of (all the possible) *input scripts* I, which can be used to redeem the associated transaction. ScriFy works by an exhaustive traversal of the space of successful traces. The trace is abandoned when an error or inconsistency in Γ is detected. Γ is satisfied by applying well-known Finite Domain Constraint Solvers. The tool uses the solver embedded in *swi-prolog*[2]. In order to better facilitate the use of ScriFy, it has been equipped with an dockered web interface[3].The open *output script* to be verified in the top text area can be inserted. We also designed a select input where it is possible to choose between several example scripts. The "Run" button triggers verification, with results reported in the bottom area. The report consists mainly of: *i)* the parsing of the given script, as a sanity check; *ii)* is the number of data required to unlock the script and the structure of the required initial stack; *iii)* the constraints on these elements, including inference of their type; *iv)* the final judgment. The redeem script checkbox, instead, is needed to insert a redemption script of a P2SH transaction. Selecting the checkbox, a new text area for the redemption script will appear, and it is possible to insert a script there.If the script inserted in the first text area (the one for the script we want to unlock) is a P2SH or P2WSH, and the checkbox is selected, the tool will verify the script, including the redeem one.

Currently, the evaluation rules and the prototype tool cover a major portion of SCRIPT's language. Interesting research for future work involves extending them further, e.g. starting with the inclusion of locktime operations, and also considering time-dependent interpretation of some commands. The overarching goal of this work, however, is to push further the automation of the verification of protocols and decentralised applications based on the Bitcoin blockchain along the lines of the atomic payment channel example.

References

[1] Stefano Bistarelli, Andrea Bracciali, Rick Klomp, and Ivan Mercanti. Towards automated verification of bitcoin-based decentralised applications. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27-31, 2023, Tallinn, Estonia.*

[2] Rick Klomp and Andrea Bracciali. On symbolic verification of bitcoin's script language. In Joaquín García-Alfaro, Jordi Herrera-Joancomartí, Giovanni Livraga, and Ruben Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International*

[2]swiprolog: `http://www.swi-prolog.org/`
[3]`http://scrify.dmi.unipg.it/`.

Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings, volume 11025 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2018.