STARK Proof for BitVM Circuit Execution

Neo Carmack
neo.carmack@gmail.com

Abstract

We arithmetize the execution trace of the BitVM circuit(represents some computation), and generate off-chain STARK proof for public verification. If the prover can not generate a correct proof, then the verifier only needs to challenge once (open one arbitrary gate) to get the BTC in the 2-sig address. If the prover can generate a correct proof, then he can get BTC by one response. The method in this paper avoids a large amount of on-chain transactions in the challenge-response process, but the amount of off-chain computation will be relatively large, mainly because generating START proof will consume more computing resources.

1 Introduction

BitVM[1] realizes Turing-complete contracts on Bitcoin and does not require a soft fork to activate new opcodes. Both parties construct tap trees according to a circuit representing some computation, and pre-sign transactions that are used in the process of challenge-response, to punish error outputs or conflict inputs(or equivocation as in the BitVM paper).

Our question is: can we detect whether Prover's computation has equivocation or incorrect output with as less as possible on-chain challenge?

Our question is: can we detect whether Prover's computation has equivocation or incorrect output with as lesser as possible on-chain challenge?

The method presented in this paper enables verifier or prover to get the BTC in the 2-signature address through one round of challenge-response.

Following the method of zk STARK[2], we arithmetic circuit execution trace, translate it into polynomials, and construct polynomial constraints for all gates and wires, so that the prover can generate ZK proof off-chain and hand it over to the verifier to verify. The process is as follows:

Verifier opens an arbitrary gate, starts the first challenge, and while waiting, asks the prover the STARK proof of his execution trace:

1. If the verification of the STARK proof succeeds, it means that the prover executed the circuit correctly and the verifier does not need to continue the challenge. After the prover responds, the verifier only needs to wait for the time lock to expire and the prover gets the BTC.

2. If the verification fails and the proof generated by the prover is wrong, then the verifier can have solid credentials and continue to on-chain challenges, forcing the prover to respond. Finally, the verifier can punish equivocation or an incorrect output of a gate.

3. If the prover does not respond to the off-chain request, nor respond to the on-chain challenge, then after the challenging Tx times out, the verifier will get the BTC.

4. If the prover does not reply to the off-chain request, but responds to the on-chain challenge, then the verifier continues the original challenge-response process (returns to 1).

There is also a possibility that the prover generates a STARK proof off-chain with the correct execution trace, but the on-chain tapleaves contain incorrect values. This situation does the prover no good: if one has the correct result, why publish an incorrect one, accepting other people's challenges and putting himself at risk?

2 Computational Integrity (CI) Statement for Contract Execution

Prover claims that the output of a certain computation is correct. BitVM converts a contract into a circuit, which is composed of a large number of gates. A gate is a basic operation, including AND, OR, NOT, NAND, etc., such as: 1 AND 0 = 0, 1 OR 0 = 1...

The output wire of the previous gate is the input wire of the next gate, thus forming a circuit. For example, the zero checking circuit [3] consists of 127 gates and 191 wires. If the input bytes are all 0, this circuit outputs true (1), Otherwise output false (0).

Prover needs to prove that he performed the computation and the output result is correct, but Verifier does not think that Prover is honest, so he needs to prove on the chain that at least one of the following problems occurred:

1. Equivocation: a wire can be both 1 and 0,

2. Incorrect output wire: for example, 1 AND 0 = 1,

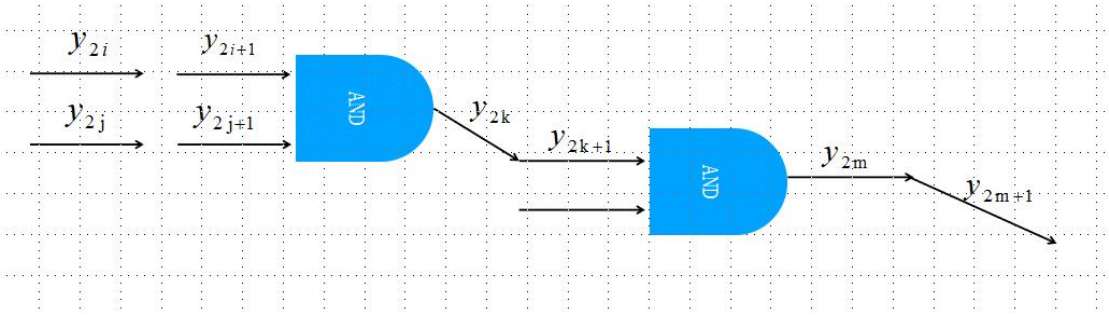
By opening gates one by one, the prover is forced to reveal the input and output values. If the above problems are identified, the verifier can obtain the BTC in the 2-sig address through an equivocation tapleave.

3 Polynomial Constraints

Let n be the total number of wires, we split every wire into two, let $\mathrm{W} = \{y_i \mid 0 \le i \le 2n\}$ be the set of all wires, then the first constraint is:

1) $\forall i < 2n : C(y_i) = y_i \times (y_i - 1) = 0$ //inputs and outputs are bits.

Output wire is y_{2k}, the input of the next gate is y_{2k+1}:



Then the second constraint is:

2) $\forall k \le n : y_{2k} = y_{2k+1}$ // there is no equivocation.

Let $G = \{g_r \mid 0 \le r \le m\}$ be the set of all gates, where $g_r = (OP, y_{2i+1}, y_{2j+1}, y_{2k})$. For any gate:

3) $\forall g_r \in G,\ y \in GF(2), i, j, k \le n : y_{2k} = y_{2i+1} \otimes (\text{or} \oplus)\ y_{2j+1}$ // all outputs are correct.

The following steps in the following sections are common in the STARK method. Readers can skip them and jump directly to the conclusion.

4 Private Polynomial for Execution Trace

All wires (with value 0 or 1) are a set of points on a two-dimensional plane:

(0, y_0), (1, y_1),(2, y_2),... (2n+1, y_{2n+1})

Use Lagrangian interpolation to find the polynomial P(x) corresponding to these points: $P(x_i) = y_i, i \le 2n+1, x_i = 0,1,2,...,2n+1, y_i \in GF(2)$. The value of function P(x) in the domain $x \in \{0,1,2,...,2n+1\}$ must satisfy the constraints 1) 2) 3) in the previous section.

For example, for constraint 1), we have:

$C(P(x)) = 0, x \in \{0,1,2,...2n+1\}$

Factoring the polynomial, we get:

$C(P(x)) = D(x)E(x)$

$D(x) = x(x-1)(x-2)...(x-2n+1)$ // the length of the trace.

Where $C(x)$ and $D(x)$ are public, and $P(x)$ and $E(x)$ are private.

5 Verification and Low Degree Test

To be added. Readers may refer to the paper [2] for details.

6 Conclusion

We chose STARK because it is quantum-resistant and has no initial trust setup. By off-chain verifying STARK proof, arbitration (challenge and force execution) becomes cheap, one to many settings would be possible.

References

[1] Robin Linus, *BitVM: Computing Anything on Bitcoin*, https://bitvm.org/bitvm.pdf

[2] Ben-Sasson, Eli, et al, *Scalable, transparent, and post-quantum secure computational integrity,* IACR Cryptol. ePrint Arch.2018 (2018): 46.

[3] Supertestnet, Tapleaf circuits, https://github.com/supertestnet/tapleaf-circuits