SatoshiVM: Advancing On-Chain Verification for Bitcoin and Enhancing Layer-2 Implementation

SatoshiVM Core Contributors

***Abstract*—This paper presents an enhanced method for validating the accurate execution of arbitrary functions on Bitcoin, specifically designed to strengthen Layer-2 (L2) block verification. SatoshiVM, a blockchain tailored for this purpose, ensures the precise execution of L2 blocks on Bitcoin without necessitating any modifications to the underlying Bitcoin protocol. It adeptly confirms the correct execution of a sequence of L2 blocks on Bitcoin, seamlessly integrating with the existing infrastructure. This capability positions Bitcoin as a secure settlement layer for L2 transactions, thereby fortifying the security of Bitcoin L2 transactions. SatoshiVM, functioning as a specialized blockchain, capitalizes on the features of Taproot, which facilitates the representation of complex function executions through a single hash. By conducting the entire zero-knowledge proof process off-chain for each block and subsequently validating the proof on Bitcoin using Taproot, SatoshiVM establishes the integrity of the blocks. Furthermore, during the submission of each block validation transaction by the L2 block miner, the corresponding block data and proof data hash are also transmitted to Bitcoin. This allows any party to utilize the submitted hash on Bitcoin to retrieve comprehensive data from a specific Data Availability layer, facilitating third-party verification of L2 blocks. The integration of SVMZK within SatoshiVM prevents transaction reordering by the sequencer, further enhancing the robustness of the proposed blockchain. This innovation contributes to advancing secure and efficient L2 solutions on Bitcoin.**

***Index Terms*—bitcoin, taproot, layer 2, SVMZK**

I. Introduction

Bitcoin, recognized as the world's pioneering decentralized cryptocurrency, has experienced exponential growth in both popularity and user adoption. However, the challenges of scalability have become increasingly evident as the user base continues to expand. The original design of Bitcoin, while groundbreaking, faces limitations in transaction processing speed and scalability. To address these challenges, the Bitcoin community is actively exploring Layer-2 (L2) solutions, aiming to enhance scalability and accommodate further growth.

L2 solutions, encompassing protocols and technologies built on Bitcoin, offer a promising avenue for more efficient transaction processing. These solutions alleviate strain on the main blockchain by redirecting a portion of the transaction volume to secondary layers. By doing so, L2 solutions significantly increase transaction processing speed, enhancing Bitcoin's capacity to handle a larger user base and promoting widespread adoption.

Among the notable L2 solutions for Bitcoin, Lightning Network [1] stands out. This off-chain payment protocol facilitates instantaneous and cost-effective transactions through bi-directional payment channels between participants. By conducting transactions off-chain, Lightning Network reduces the load on the main blockchain, enabling near-instantaneous transactions with minimal fees. Widely embraced within the Bitcoin community, Lightning Network holds the potential to reshape the future growth of Bitcoin.

Another noteworthy L2 solution is Liquid Network [2], developed by Blockstream. This federated sidechain enables faster and more confidential Bitcoin transactions. Liquid Network supports the creation of pegged assets, facilitating the transfer of assets between different exchanges and enhancing liquidity. As a scalable and secure platform, Liquid Network contributes to more efficient Bitcoin transactions, supporting the expansion of both businesses and individuals.

With the ever-growing user base, scalability has become a pressing concern for Bitcoin. L2 solutions such as Lightning Network and Liquid Network offer promising approaches to address these scalability challenges. By offloading transactions to secondary layers, these solutions enhance Bitcoin's transaction processing capacity, ensuring sustained growth and adoption. The active exploration and implementation of L2 solutions within the Bitcoin community demonstrate a commitment to overcoming scalability limitations and supporting Bitcoin's continued expansion.

This paper proposes a novel method for verifying the correct execution of arbitrary functions on Bitcoin without relying on the challenge-response protocol utilized in other solutions. Our improvement scheme significantly streamlines the L2 block verification scheme's implementation on Bitcoin, focusing on enhancing the bit commitments component of the original BitVM Tapleaf Circuit scheme [3] [4]. This is achieved by measuring the time difference between the bit commitments and verification transactions and considering their combined results. Through this approach, we reduce the necessary two transactions, the funding transaction, and the anti-contradiction transaction, in the original design, leading to increased efficiency. Moreover, it facilitates independent third-party observation of the final results. In the presence of an error in the submitted execution path, the verifier can always identify a spendable leaf script. The verifier executes functions off-chain using preimages provided by the prover and locates the error. This scheme requires only two Taproot transaction during the verification process.

II. Architecture

The architecture of our proposed L2 block verification system involves the Prover employing integrated circuit design programs such as SCALE-MAMBA to transform the verify function into a specific circuit form, typically represented in

Bristol format. In this design, each possibility of every line in the circuit is treated as a leaf of the Taproot. Consequently, all potential values of unlocking scripts for each step in the execution process of the entire circuit can be utilized to spend the Unspent Transaction Output (UTXO).

The verification process is facilitated by the verifier, who can identify execution errors by pinpointing the faulty leaf and subsequently spend the funds as evidence of the mistake in execution. This approach ensures transparency and accountability in the L2 block verification process.

The complete circuit representation of a function can be extensive; for instance, a 64-bit adder may consist of over 400 gates. To streamline demonstration and enhance comprehension, we propose using a simplified circuit as an illustrative example. This allows for a more accessible presentation of the verification process, showcasing the functionality of the L2 block verification system.

A. Circuit

As illustrated in Figure 1, the presented circuit adheres to Bristol format. The initial line specifies the essential count of "gates" and "wires" for the circuit, signifying 4 gates and 7 wires in this particular instance. Subsequently, the second line delineates the quantity of "input" wires contributing input bits to the function. In this case, there is a singular input to the function, comprising 3 bits. Additionally, it denotes the number of "output" wires emerging from the circuit.

```
4 7
1 3
1 1
1 1 0 1 INV
2 1 1 3 4 AND
1 1 4 5 INV
2 1 2 5 6 AND
```

Fig. 1: Bristol format representation of a circuit.

Further details encompass the circuit's operational instructions. The first element designates the count of input wires, while the second element represents the number of output wires. Subsequent elements, guided by the preceding parameters, indicate the specific input and output wires. Lastly, the instructions detail the operations associated with each wire and gate. A visual representation of the corresponding gate circuit diagram for this configuration is available in Figure 2.

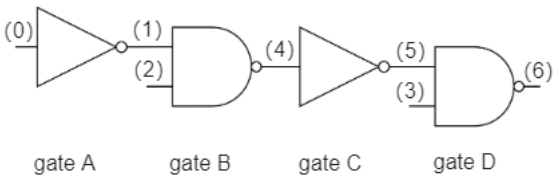

Fig. 2: Circuit representation through logic gates.

B. Preimage Commitment

The application of preimage commitment assumes a pivotal role in concealing the internal wiring structure of a circuit until the execution path is disclosed to the verifier. Following the prover's disclosure of commitments linked to inputs and outputs, the verifier is empowered to scrutinize the execution trace based on the provided preimages. Consideration of Figure 3 sheds light on this matter, depicting the third NOT gate (gate C) as an illustrative example with its input wire designated as 4.

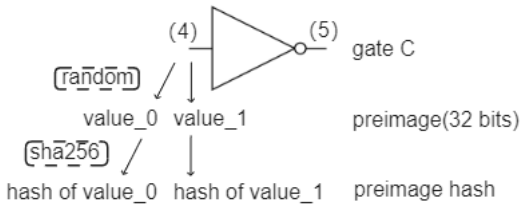

Fig. 3: Commitment of wire4 in the NOT gate.

The prover, in this context, randomly generates two 32-bit numbers representing scenarios where the input is 0 and 1, respectively. The numerical representations will be disclosed to the verifier when the prover provides the execution trace of the function. The verifier, armed with this input, along with the script provided in Figure 4, can evaluate the correctness of the gate execution. The prover generates a preimage hash through the hashing process applied to the preimage. All preimage hashes are then conveyed to the verifier. Collaboratively, a Taproot tree is constructed employing these values, with each leaf encapsulating the entirety of potential execution inputs.

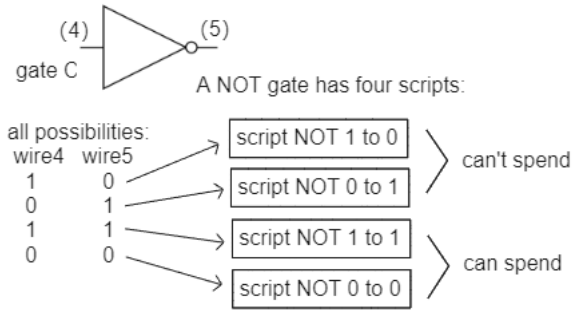

Fig. 4: Potential execution diversity: a NOT gate with four input-output combinations and an AND/XOR gate with eight possibilities. The number of scripts corresponds to the variability in potential execution processes. The prover's preimage facilitates correctness verification through script execution by the verifier.

C. Scripts and Taproot

Our inventive framework is established through the measurement of the time gap between bit commitments and verification transactions, taking into account the collective outcomes they produce. In this section, we enumerate the primary core operations, namely, the scripts for bit commitments

and verification Taproot transactions. Additionally, the time-locked leaf plays a crucial role in the on-chain function and time difference composite evaluation. While its significance is paramount in our innovation, its script is more generic. Therefore, we refrain from providing an extensive elaboration in this section and will delve into its specific functionalities in subsequent sections.

Figures 5 and 6 demonstrate the scripts involved in the bit commitments transaction, showcasing, respectively, the specific script ensuring the prover submits the preimage of a specific wire and the full script ensuring the prover submits preimages for all execution paths.

```
Inputs:
input_0
input_1
...
input_n

Scripts:
script_0
script_1
...
script_n
<Prover's Public Key>
OP_CHECKSIG
```

Fig. 5: One script of bit commitments Taproot.

Figure 6 illustrates a pivotal subleaf within the bit commitments transaction. Another time-locked leaf is configured to allow the verifier to spend the corresponding UTXO after a set period of 10 blocks (any value earlier than the verification Taproot expiry time suffices).

```
input_i:
<preimage of a value of wire_i>

script_i:
OP_SHA256
<preimage hash of value 0 of wire_i>
OP_EQUAL
OP_SWAP
OP_SHA256
<preimage hash of value 1 of wire_i>
OP_EQUAL
OP_BOOLOR
OP_VERIFY
```

Fig. 6: Segment of one bit commitments Taproot script.

This leaf establishes a condition that mandates the complete acceptance of the execution path before the prover can reclaim the funds. This ensures that the prover cannot manipulate the verification Taproot to default through the refusal to provide preimages as a means of cheating. The outcomes of such determinations constitute crucial criteria for the comprehensive assessment.

Figure 4 reveals that a NOT gate encompasses four potential input-output combinations, constituting distinct execution scenarios for the prover. Within these scenarios, two instances signify correct execution of the NOT logic gate, preventing the verifier from spending UTXO. Conversely, two cases depict incorrect execution, allowing the verifier to spend UTXO and detect the prover's error. In contrast, both AND gate and XOR gate offer eight possible scenarios each, collectively forming the leaves of Taproot.

Figure 7 serves as an illustrative script example designed to ascertain the prover's accurate execution of a NOT gate. This script relies on the preimages of wire4 and wire5 values. The script functions correctly only when the prover supplies the preimage corresponding to the value 0 for wire4 and the value 1 for wire5. This ensures proper execution and affirms the correctness, thereby inhibiting the verifier from initiating expenditure. Three alternative scripts for this NOT gate exist, wherein if the prover provides preimages resulting in the same input and output values of either 0 or 1, the verifier can spend it, revealing any inaccuracies in the prover's results.

```
Inputs
<preimage of a value of wire_4>
<preimage of a value of wire_5>

Scripts
OP_TOALTSTACK
OP_SHA256
<preimage hash of value 0 of wire_4>
OP_EQUALVERIFY
OP_0
OP_NOT
OP_FROMALTSTACK
OP_SHA256
<preimage hash of value 1 of wire_5>
OP_EQUALVERIFY
OP_1
OP_NUMNOTEQUAL
OP_VERIFY
<Verifier's Public Key>
OP_CHECKSIG
```

Fig. 7: One possible script of verification Taproot.

D. Verification Taproot

Figure 8, gate B and gate C form a pair of interconnected logic gates. Gate B, an AND gate, exhibits eight possible scripts, yielding four outputs of 0 and four outputs of 1. Gate C, a NOT gate, features four potential inputs—two being 0 and the other two being 1. In our enhanced scheme, we treat the preimage corresponding to the scenario where the output of gate B is consistently 0 as equivalent to the preimage corresponding to the scenario where the input of gate C is

always 0. Similarly, this equivalence applies to the scenario where the output of gate B is consistently 1 and the input of gate C is always 1. Consequently, upon the prover providing complete preimages, the verifier can identify inaccuracies by examining the consistency of the script's input and output.

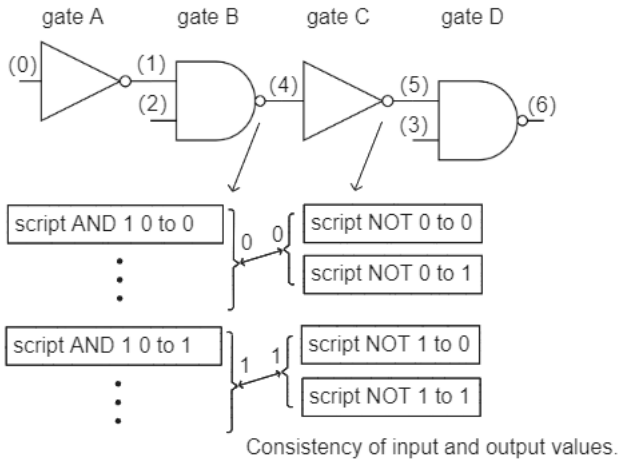

Fig. 8: Association of connected values for wire4: using wire4 as an illustrative example, our improvement plan establishes an association between the connected values, ensuring they share the same preimage. This synchronization is crucial, as only when the output of gate B aligns with the input values of gate C can both gates maintain coherence simultaneously.

In the innovative scheme proposed in this paper, the verification of potential cheating by the prover requires only two Taproot transactions. The bit commitments Taproot ensures that, upon the prover's submission of preimages, the verifier can authenticate the leaf scripts on the verification Taproot based on this data. Taking gate C in Figure 9 as an example, suppose the prover attempts malfeasance by providing an input of 0 to gate C when the output of gate B is 1. Regardless of the value assigned to wire5, the verifier possesses the capability to spend the erroneous script in gate C. Consequently, there is no need for an anti-contradiction mechanism to prevent the prover from presenting conflicting preimages.

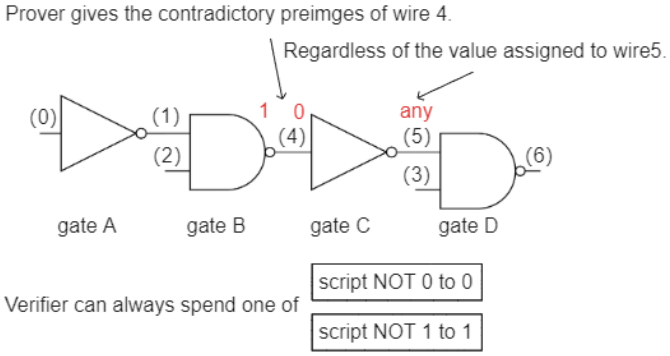

Fig. 9: Determining the legitimacy of function execution with two Taproot transactions.

E. Framework Visualization

The flowchart in Figure 10 illustrates the procedural diagram of our enhanced scheme. Leveraging the time difference between the two transactions' time locks, the prover and verifier can independently and non-interactively complete the entire function verification process. Additionally, third parties can infer the execution details of the function based on the combined results of the two transactions.

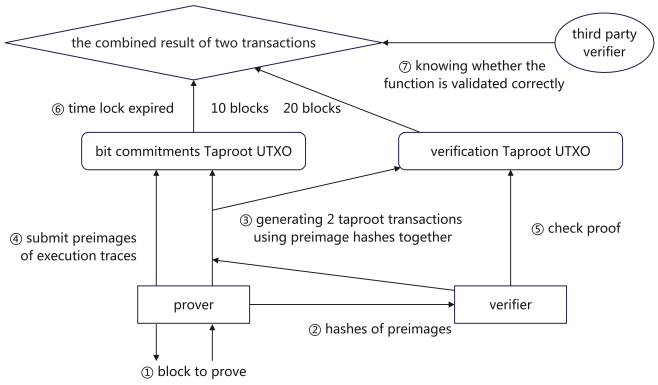

Fig. 10: Procedural diagram of the enhanced scheme.

To illustrate the entire mechanism more clearly, we additionally provide a chronological verification process of Prover and Verifier in Figure 11.

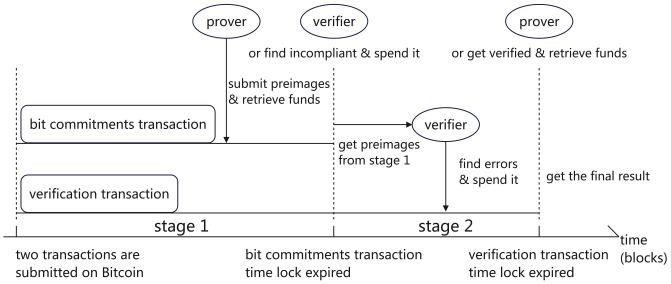

Fig. 11: Chronological verification process of Prover and Verifier on Bitcoin.

F. Third Party Verification

In this section, we will elucidate why a third party can ultimately verify the correctness of a function execution. As illustrated in Table I, if the verifier spends the bit commitments transaction, it indicates that the prover did not submit compliant preimages, leading to a failed verification of the function execution. If the prover spends the bit commitments transaction, the subsequent analysis focuses on who spends the verification transaction. If the verifier expends the verification transaction, it substantiates that the prover provided an incorrect execution. Conversely, if the prover expends the verification transaction, it signifies that the verifier cannot identify a script to spend, thereby proving the correctness of the prover's execution.

bit commitments txn	verification txn	execution result
spent by Verifier	/	wrong
spent by Prover	spent by Verifier	wrong
spent by Prover	spent by Verifier	correct

TABLE I: Integrated results of time difference transactions.

G. Bitcoin Layer-2

Through these enhancements, we can elevate the verification speed of a function and mitigate costs on Bitcoin. Consequently, it can offer crucial support for Bitcoin L2s. Figure 12 illustrates an on-chain verification scheme for Bitcoin rollups.

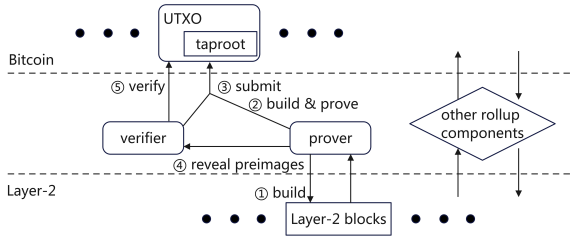

Fig. 12: A Bitcoin L2 based on an on-chain verification system.

III. SVMZK: Anti Transaction Reordering

A. Overview

While the challenge of verifying L2 blocks on Bitcoin has been addressed, there remains a potential issue where sequencers may submit incomplete blocks, a concern common in traditional blockchains, Transaction Reordering. In the L2 context, the singular nature of a sequencer amplifies the likelihood of selectively submitting or withholding specific transactions. In L2 solutions built on the Ethereum Virtual Machine (EVM), verifiers or validators actively engage in the L2 peer-to-peer network, overseeing the sequencer (or batcher) within the smart contract to ensure the integrity of L2 blocks.

However, the absence of smart contracts within the Bitcoin ecosystem necessitates the deployment of a bespoke solution fostering symbiosis between the sequencer and verifier, thereby ensuring the veracity of submitted blocks. This specialized approach, harmoniously integrated with SVMZK, an efficient NIZK proof toolkit, facilitates the streamlined submission of concise results to Bitcoin. Simultaneously, it serves as a safeguard against potential collusion between the sequencer and validator, thereby preempting the generation of invalid outcomes.

As illustrated in Figure 13, this collaborative procedural framework necessitates active participation from both the sequencer and validator. Preliminary commitment submissions of data from their respective mempools serve the pivotal function of precluding subsequent modifications to input parameters. Subsequently, leveraging the original data, both entities generate proofs. Ultimately, in conjunction with a verifier, an on-chain confirmation process ensues, validating both the overall verification process and the subsequent unveiling process facilitated through Taproot.

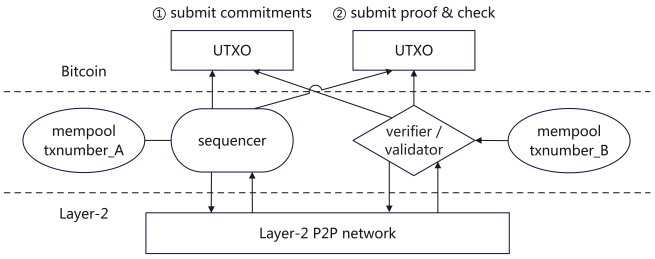

Fig. 13: A Bitcoin L2 prevents the issue of transaction reordering.

The primary approach for verifying the accurate packaging of transactions by the sequencer involves a comparison of the transaction count in the mempool of both the sequencer and the validator. This comparison is facilitated through the utilization of the Subtraction and Comparison algorithms provided by SVMZK. The corresponding details are expounded in the forthcoming manuscript.

B. Manuscript

1) Notations: Let $\mathbb{G}$ be a cyclic group with prime order q, and denote g as the generator. Denote $[a; r]_{\mathsf{ck}}$ as the Pedersen commitment of $a \in \mathbb{Z}_q$ under the commitment key $\mathsf{ck} := (g, h)$ using randomness $r \in \mathbb{Z}_q$, i.e., $g^a h^r$. When ck is clear, and/or r is not important in the context, we use $[a]$ for simplicity.

2) Subtraction: We utilize the Schnorr protocol to prove that $\{a-b, \alpha-\beta\colon [c] = g^{a-b}h^{\alpha-\beta}\}$. Applying the Fiat-shamir transformation, it can be easily made into a non-interactive version.

SVMZK for $a - b = c$

Statement:

- $[a] := g^a h^\alpha, [b] := g^b h^\beta$ and $[c]$ such that $[a]/[b] = [c]$;

Witness:

- $a - b, \alpha - \beta$ such that $[c] = g^{a-b}h^{\alpha-\beta}$;

Prover:

- Randomly select $x, y \leftarrow \mathbb{Z}_q$;
- Compute:
  - $d = g^x h^y$,
  - $e = \mathsf{hash}(g, h, [a], [b], [c], d)$,
  - $u = x + (a - b)e$,
  - $v = y + (\alpha - \beta)e$;
- Send $\pi = (d, u, v)$;

Verification:

- Compute $e = \mathsf{hash}(g, h, [a], [b], [c], d)$;
- Output 1 iff the following conditions hold:
  - $[c] = [a]/[b]$,
  - $d \cdot [c]^e = g^u h^v$;

Fig. 14: SVMZK for $a - b = c$

3) Comparison: We assume that the values $x \in [-2^\ell, 2^\ell]$, e.g., $\ell = 31$, so we can decompose x into $b_0, ..., b_{\ell-1}$ and a sign bit b_ℓ (1 for negative and 0 for non-negative) such that $x = (-1)^{b_\ell} \cdot \sum_{i=0}^{\ell-1} b_i \cdot 2^i$. Given $[a]$,$[b]$, to show that $a \geq b$ is equivalent to show $[a]/[b]$ commits to $a - b \geq 0$. If we want to prove that $x \geq 0$, it is equivalent to prove $x = \sum_{i=0}^{\ell-1} b_i \cdot 2^i$ and $\forall i \in \{0, ..., \ell - 1\}, b_i(1 - b_i) = 0$ in $\mathbb{Z}_q$.

SVMZK for $x \geq 0$

Statement:

- $[x] = g^x h^s$;

Witeness:

- $x, s \in \mathbb{Z}_q$;

Prover:

- Decompose x into $b_0, ..., b_{\ell-1}$ such that $x = \sum_{i=0}^{\ell-1} b_i \cdot 2^i$;
- For $i \in \{1, ..., \ell - 1\}$, pick random $r_i \leftarrow \mathbb{Z}_q$ and commit $[b_i] = g^{b_i} h^{r_i}$;
- Set $[b_0] = [x] / \prod_{i-1}^{\ell-1} [b_i]^{2^i}$;
- Randomly select $u', b'_0, ..., b'_{\ell-1}, r' \leftarrow \mathbb{Z}_q$;
- Compute
  - $d_1 = g^{\sum_{i=0}^{\ell-1} b'_i} h^{r'}, d_2 = g^{\sum_{i=0}^{\ell-1} b_i b'_i} h^{u'}$,
  - $e = \mathsf{hash}(g, h, [x], [b_0], ..., [b_\ell - 1], d_1, d_2)$,
  - $\hat{u} = \sum_{i=0}^{\ell-1} (e^i - \hat{b}_i) \cdot r_i + u', \hat{r} = \sum_{i=0}^{\ell-1} e^i \cdot r_i + r'$,
  - For $i \in \{0, ..., \ell - 1\}$: $\hat{b}_i = e^i \cdot b_i + b'_i$;
- Send $\pi = ([b_1], ..., [b_{\ell-1}], d_1, d_2, \hat{u}, \hat{b}_0, ..., \hat{b}_{\ell-1}, \hat{r})$;

Verification:

- Compute
  - $[b_0] = [x] / \prod_{i=1}^{\ell-1} [b_i]^{2^i}$,
  - $e = \mathsf{hash}(g, h, [x], [b_0], ..., [b_{\ell-1}], d_1, d_2)$;
- Output 1 iff the following conditions hold:
  - $d_1 \cdot \prod_{i=0}^{\ell-1} [b_i]^{e^i} = g^{\sum_{i=0}^{\ell-1} \hat{b}_i} h^{\hat{r}}$;
  - $d_2 \cdot \prod_{i=0}^{\ell-1} [b_i]^{e^i - \hat{b}_i} = h^{\hat{u}}$;

Fig. 15: SVMZK for $x \geq 0$

IV. Conclusion

In summary, our proposed advancement in the verification methodology of arbitrary functions on Bitcoin, leveraging enhancements to BitVM [3] and Tapleaf Circuits [4], introduces an intricately designed protocol. This protocol, which strategically associates preimages of wires, not only accelerates the verification process but also offers a cost-effective solution. This augmentation is tailored to fulfill the rigorous block verification prerequisites of Layer-2, wherein Bitcoin serves as the foundational network. Furthermore, the method incorporates essential measures to prevent transaction reordering by the sequencer, a functionality achieved through the implementation of SVMZK.

It is imperative to acknowledge that the current framework calls for continued refinement, particularly in the realms of script storage and the facilitation of off-chain data sharing. Future endeavors should focus on addressing these areas to fortify the overall robustness and scalability of the proposed verification system.

References

[1] J. Poon and T. Dryja. (2016) The bitcoin lightning network: Scalable off-chain instant payments. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[2] J. Nick, A. Poelstra, and G. Sanders. (2020) Liquid: A bitcoin sidechain. [Online]. Available: https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf

[3] R. Linus. (2023) BitVM: Compute Anything on Bitcoin. [Online]. Available: https://bitvm.org/bitvm.pdf

[4] supertestnet. (2023) Tapleaf Circuits: A proof-of-concept implementation of BitVM for bristol circuits. GitHub Repository. [Online]. Available: https://github.com/supertestnet/tapleaf-circuits